Analysis of Transformations to Socio-Technical Systems Using Agent-based Modeling and Simulation

A Thesis
Presented to
The Academic Faculty

by

Anuj P. Shah

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Industrial and Systems Engineering

Georgia Institute of Technology
May 2006

Analysis of Transformations to Socio-Technical Systems Using Agent-based Modeling and Simulation

Approved by:

Dr. Amy R. Pritchett, Advisor
School of Aerospace Engineering, and
School of Industrial and Systems Engineering
Georgia Institute of Technology

Dr. Kevin Corker,
Industrial and Systems Engineering Department
San Jose State University

Dr. William B. Rouse,
School of Industrial and Systems Engineering
Georgia Institute of Technology

Dr. Ashok Goel,
College of Computing
Georgia Institute of Technology

Dr. David M. Goldsman
School of Industrial and Systems Engineering
Georgia Institute of Technology

Date Approved: April 10, 2006

ACKNOWLEDGEMENTS

This thesis would not have been possible without the support and guidance of my advisor, Dr. Amy R. Pritchett. My deepest thanks to her for her patience, mentoring, and encouragement over the years; she pushed me to do things I did not believe I could. I will always remember the long and insightful conceptual discussions we have had throughout the course of my Ph.D.

I would also like to extend my gratitude to Dr. Kevin Corker, Dr. William B. Rouse, Dr. Ashok Goel and Dr. David M. Goldsman, for taking the time and effort to serve on my thesis committee. Dr. Corker's support for my research project and his insights with respect to the field of Human Machine Systems were instrumental to the success of this thesis. Dr. Rouse's insights from a Systems Engineering perspective, Dr. Goel's deep interest in understanding and clarifying the concepts developed in my dissertation, and Dr. Goldsman's knowledge of simulations proved invaluable to the completion of this thesis.

I would also like to thank my parents for their continued emotional support through all my endeavors. I am also very thankful to my wife, Prachi, for her love and support that gave me the strength to continue through the stressful stages of my work.

Last but not the least, I want to thank all my colleagues and friends, who have kept life at Georgia Tech interesting.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Transformations to socio-technical systems may be enacted through changes in technology, processes, information, workers and their organization. Typically, these changes are implemented by changes within a component (e.g., changing a technology or procedure) or by changing the interrelation between components (e.g., changing which workers have access to specific information), yet the measures of system performance are not measured at the level of the affected components but instead measured as changes to the overall emergent functioning of all the components taken together. This thesis established a conceptual framework and a simulation platform for modeling and simulation of socio-technical systems for a priori computational analysis of the impact of such transformations.

This thesis builds on the principles of cognitive engineering to describe the components of the work environment, i.e., technology, processes and information, in work relevant ways and using a structure-preserving model, i.e., a model form that describes their aspects using the same attributes and structure as used by system designers and operators. This thesis also builds on the principles of agent-based modeling to model workers and their interactions with the work environment. These models are specified through a conceptual framework that includes both declarative models describing which components are included within the system and their interrelations, and computational models of those complex, dynamic behaviors that cannot be adequately described declaratively. Declarative modeling enables easy composition and modification of component models; in addition, by computationally assembling all required components collectively enables automatic generation of a declarative model of the system, which can

be analyzed for network dependencies. To facilitate declarative modeling and network analysis, this thesis both established an XML representation for the declarative models, and developed a mechanism that then automatically assembles, from the individual components' specifications and interrelations, a network-level model of the entire system in XML which can serve to analyze network dependencies between components. Likewise, this thesis developed an object-oriented modeling framework in which complex, dynamic internal behaviors are each encapsulated as computational objects which can model the complex behaviors of system components and workers using any of a wide range of model structures as appropriate.

The combination of the declarative and computational models also enables computational simulations to predict the system performance that will emerge from a network of components when placed in a given scenario. Thus, this thesis also developed an agent-based simulation platform that can simulate the collections of worker and work environment models created using the conceptual framework for the purpose of operational analysis.

The theoretical contributions of this thesis include the conceptual framework as a broadly applicable and structure-preserving representation of the important aspects of socio-technical system behavior, and associated extensions to cognitive engineering descriptions of the work environment. These insights, combined with the simulation platform, also enable computational modeling, analysis and prediction of socio-technical system performance with a comprehensiveness and detail not possible before. The theoretical and practical utility of these developments is demonstrated through a case study in air traffic control.

CHAPTER 1

INTRODUCTION

1.1 Socio-technical Systems

In general a socio-technical system is any system that is made up of people, technologies, processes and information, and that requires successful integration of all these elements for its proper functioning. For example, the air traffic control system in Figure 1 consists of physical components such as aircraft (pink dots in figure) that fly through the airspace to their respective destination airports, workers such as the air traffic controllers that manage the air traffic, the controllers' sectors (ZLA-39, ZLA-37, ZLA-20, ZLA-19 and SCT-FDR) that spatially distribute the airspace, technological components such as radar and communication radios, and the air traffic control procedures which each controller is allowed to use (e.g., conflict avoidance, miles-in-trail metering, and time-based metering). All these elements must be integrated and coordinated for successful operation of this system.

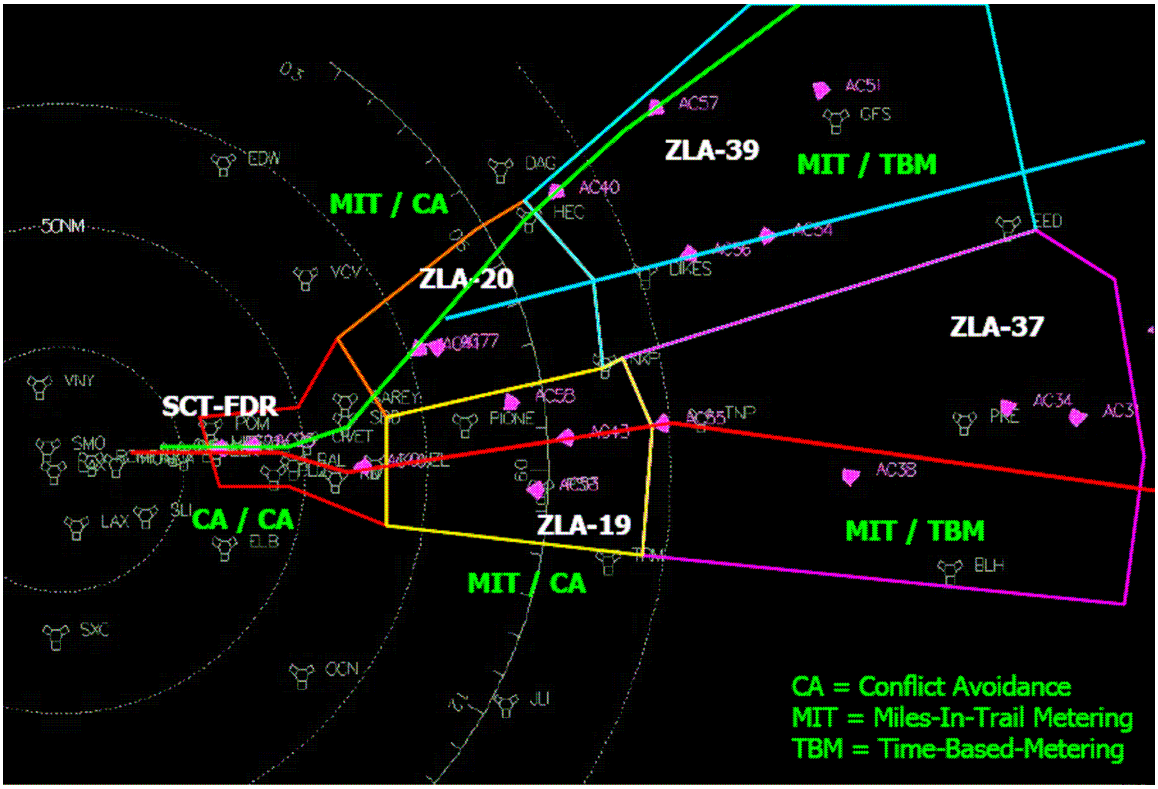

Figure 1: Example of a socio-technical system – an air traffic control system

In particular, this thesis focuses on those systems where the dynamics of interest are the work (i.e., the purposeful cognitive and physical activities) of the workers and the system performance *emerging* from their collective work. For example, in the air traffic control system in Figure 1 one measure of system performance is the rate at which the aircraft can land at a given airport, maximized when arriving aircraft are spaced as close as possible within the safety limits on distance between any two aircraft. Such performance results through collective and coordinated work of the air traffic controllers managing the approach sectors and the pilots flying the aircraft, and cannot be credited to any individual.

Not only does system performance emerge from collective work of the workers, their work also changes both the state and the structure of the system through time. Furthermore, the work environment shapes the behavior of the workers. For example, there are intrinsic physical limitations on the maximum acceleration and deceleration rate of the aircraft, thus requiring air traffic controllers to plan ahead and shaping their choice of actions. Such a two-way relationship between the workers and the work environment, and the emergence of system performance from their interaction, requires modeling both these elements of the system and their work-relevant interactions such that their localized interactions can manifest the global emergent behavior.

1.2 Transformations in Socio-technical Systems

Most modern socio-technical systems are transformed through changes in the workers, the work environment, and/or their work-relevant interactions. To develop a deeper understanding of the nature of these transformations, let us look at a few examples. One kind of transformation can be enacted at the 'network' level by changing the configuration of the system in terms of which components constitute it and how they relate to each other. The example in Figure 1 shows an air traffic control system where the airspace is spatially distributed into five sectors, namely: ZLA-39, ZLA-37, ZLA-20, ZLA-19 and, SCT-FDR. Each of these sectors is controlled by their respective air traffic controller, who is given specific air traffic control procedures. This system can be transformed at the network level by changing which procedures are given to each air traffic controller, such as providing time-based-metering procedures instead of miles-in-trail procedures.

Similarly, transformations can be enacted at the 'component' level when the internal dynamics of a component may be changed, reflecting a change in its behavior and interaction with other components in the system, thus also impacting the system level performance. Figure 2 provides an example of a physical change to an aircraft by modifying the algorithms used by its flight management system to resume course after it has been steered away. Component level changes can also be made in non-physical components; for example, a work-process such as an air traffic control procedure can be changed by changing the specification of activities, their ordering or their defining parameters (Figure 3).

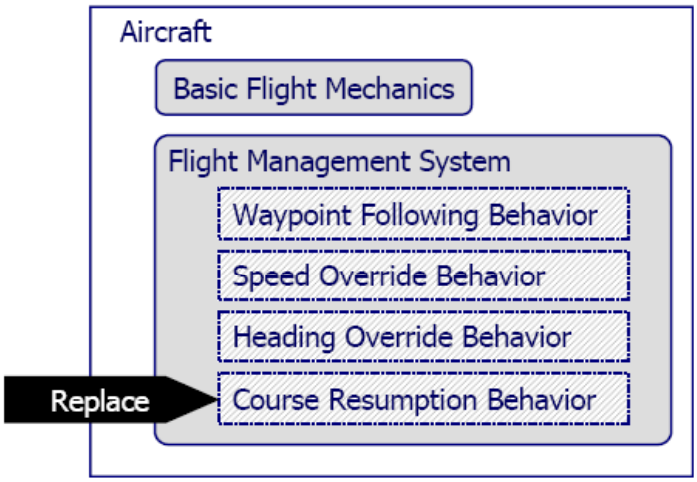

Figure 2: Example of a component-level transformation in a physical component

Process to vertically separate two arrivals

(a) Hold altitude of higher aircraft

(b) Recheck every 30 seconds if resuming course is conflict free

(c) Resume course when appropriate

Transformed process to vertically separate two arrivals

(a) Hold altitude of higher aircraft until vertical separation is achieved

(b) Once separated vertically, descend higher aircraft to next waypoint altitude at vertical speeds less than that of lower aircraft. When assigning vertical speed, compensate for the following

  (1) If true airspeed of aircraft is greater than 250 Knots, assign 2000 feet per minute

  (2) If true airspeed of aircraft is less than 250 knots, assign 1500 feet per minute

(c) Recheck every 30 seconds if resuming course is conflict free

(d) Resume course when appropriate

Figure 3: Example of a component-level transformation in a work-process component

Whereas the above examples discussed changes in the work environment, transformations can also be enacted through changes in worker(s), i.e., at the 'worker' level. For example, the contents of the long-term memory or the intrinsic skills of a worker could be changed through training. Such a change will affect the behavior of the worker(s) and result in a system transformation.

While these transformations are enacted at fairly "low" levels of abstraction, their motivation is an increase in system performance as assessed at a system level of abstraction. For example, whereas a transformation maybe enacted at the component level by changing an air traffic controller's display, or in processes by introducing a new kind of coordination between the pilots and the controllers, or at the network level by removing the controller altogether and requiring the pilots to separate themselves, the

motivation is improving safety and throughput of the airspace. Due to the obvious complexity of mapping between these two levels, i.e., the complexity of explaining emergent performance in terms of the low level factors and conversely predicting the effect in system level performance from changes in the low level factors, in most cases the choice of a particular transformation for achieving some specific system level effect is based on the intuitions and extrapolations of field experts. To this, a rigorous explanatory and predictive capability will help lend more credibility, objectivity and comprehensiveness in analysis.

1.3 Problem Statement

This thesis seeks to explain and predict, through two types of analysis, the component, network and worker level socio-technical system transformations discussed in the preceding section. First, *operational analysis* examines the evolution of system state through time and the emergence of system performance from work-relevant interactions of the workers and the work environment. Second, *network analysis* examines the dependencies between the parts of the system, i.e., dependencies that result from the arrangement of work-relevant interrelations between the parts. Operational analysis is concerned with performance and behavior over time; network analysis is conducted on a snapshot of the system at any given point in time.

For providing a capability to perform these two analyses, based on §1.1, the workers, their work environment, and their work-relevant interactions need modeling. Furthermore, based on §1.2, these models should be able to appropriately represent the

component, worker and network level transformations. This thesis defines the scope of this modeling problem by specifically focusing on socio-technical systems where:

1. Work of the workers, and system performance emerging from that work, are the dynamics that interest the analyst;
2. The workers may have individual goals;
3. The workers and other system components interact in work-relevant ways; and
4. System performance, measured relative to system objectives, emerges from the collective interactions and individual actions of workers and other components.

Furthermore, this thesis focuses on system transformations that are enacted by changes of these three types:

1. *Worker level*: changes to the intrinsic characteristics of the workers, such as their skills;
2. *Component level*: changes intrinsic to the physical, technological and process components of the work environment; and
3. *Network level*: changes to which components comprise the system and work-relevant relationships between them.

This thesis' intention behind developing this modeling and analysis capability is to inform the design of transformations in socio-technical systems. In particular, this thesis aims to develop a conceptual framework and simulation platform for designers to test and iteratively refine their intuitions about transformations. This thesis, therefore, addresses several systems engineering requirements for analysis using this thesis' conceptual framework and simulation platform:

1. Must be *explanatory* and *predictive*, so that substantial and/or revolutionary transformations can be examined a priori to prototype development, system implementation, and large-scale longitudinal studies.
2. Should be *computational*, because of the specificity and the objectivity it brings, and the practicability of leveraging the computational power of computers for conducting a large number of 'what-if' evaluations and for analyzing transformations to large-scale systems.
3. Should be *structure-preserving*, i.e., models describing the system, its components and their interrelations should use the same attributes as used by system designers and operators. Use of structure-preserving models should alleviate the modeling complexity often encountered when using models that require significant translation from designers' and operators' descriptions of the system. Furthermore, structure-preserving models allow the designers and analysts to iterate through the design evolution process in close correspondence with real world iterations, thus developing a more realistic mapping between system performance and transformation variables as the design evolves.
4. Should closely correspond with and provide for common system's engineering processes such as spiral design.

1.4 Summary of the Proposed Solution

In this thesis, a novel and domain-independent conceptual framework and a simulation platform have been developed to enable the analyses discussed in the preceding sections. As defined in Table 1, the conceptual framework provides the basis for domain- and

application-specific computational models, and the simulation platform enables specific models to be simulated. The conceptual framework serves to address the issues of modeling socio-technical systems and the corresponding approach to analyzing transformations in them. The simulation platform serves as a tool for system designers and addresses the practical systems engineering issues of computationally analyzing large-scale systems.

Table 1: Defining 'Framework', 'Platform', 'Model' and 'Simulation'

Independent of Specific Application	**Specific to an Application**
Conceptual Framework: A specification of core principles and constructs specified to the level of detail that enable computational modeling of the socio-technical systems and their components.	Computational Model: The constructs of the conceptual framework represented as computer programs or mathematically to apply to specific situations or types of situations.
Simulation Platform: Software and computational implementation of the main modules and mechanisms underlying the conceptual framework, and the general mechanisms that employ this implementation to evolve a system model's state through time.	Simulation: A simulation is an imitation of some real thing, state of affairs, or process. In the context of this thesis, a simulation is an evolution of system model's state through time.

To accomplish operational analysis, first, this thesis has developed a conceptual framework in which a modeler can computationally describe and capture work-relevant relationships amongst the components of the work environment (Figure 4). Workers are assumed to accomplish their work through the application of problem solving approaches that employ these relationships. These problem-solving approaches and any relevant limitations and capabilities of workers are encapsulated in agent-based models based on cognitive models and human performance models, or through more mechanical formulations from decision theory or artificial intelligence. Agent-based models represent the autonomy, interactivity, and the discrete and localized activity of workers, from which the system behaviors emerge. Finally, this thesis has developed a simulation platform that can collectively simulate these models of the work environment and of the agents.

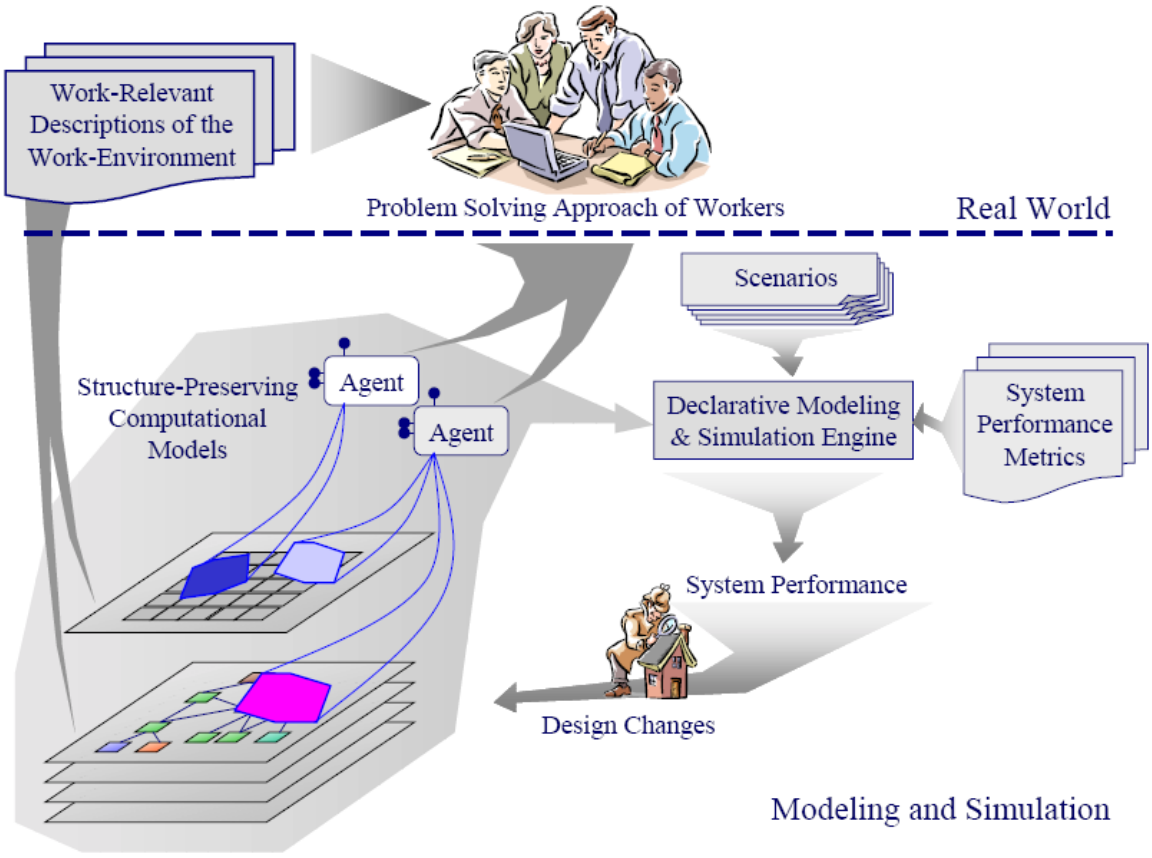

Figure 4: This thesis' approach to modeling and simulating socio-technical systems for operational analysis

To accomplish network analysis, this thesis uses the same conceptual framework, which was purposefully constructed to specify the models of the components, worker and their interrelations declaratively such that their individual network-relevant attributes can be modified individually and locally. Instances of those models that make a particular system are collectively fed to a declarative model construction engine that assembles the network-level model of the system by fitting together these individual elements based on

their interrelations (Figure 5). Once constructed, the network-level model can be queried to identify dependencies between the components.

This model construction engine is integrated with the simulation platform to account for changes in the network as a simulation advances, as the system composition and the component interrelations may change due to the work of the workers. At any point in time in the simulation, the declarative specification of this dynamically changing network can be queried for network level dependencies in the system. These dependencies can be related to the current state of the simulated system and may be intuitively or logically related to the past evolution of system state, thus affording useful analysis of the impact of transformations on the evolution of the structure of the system.

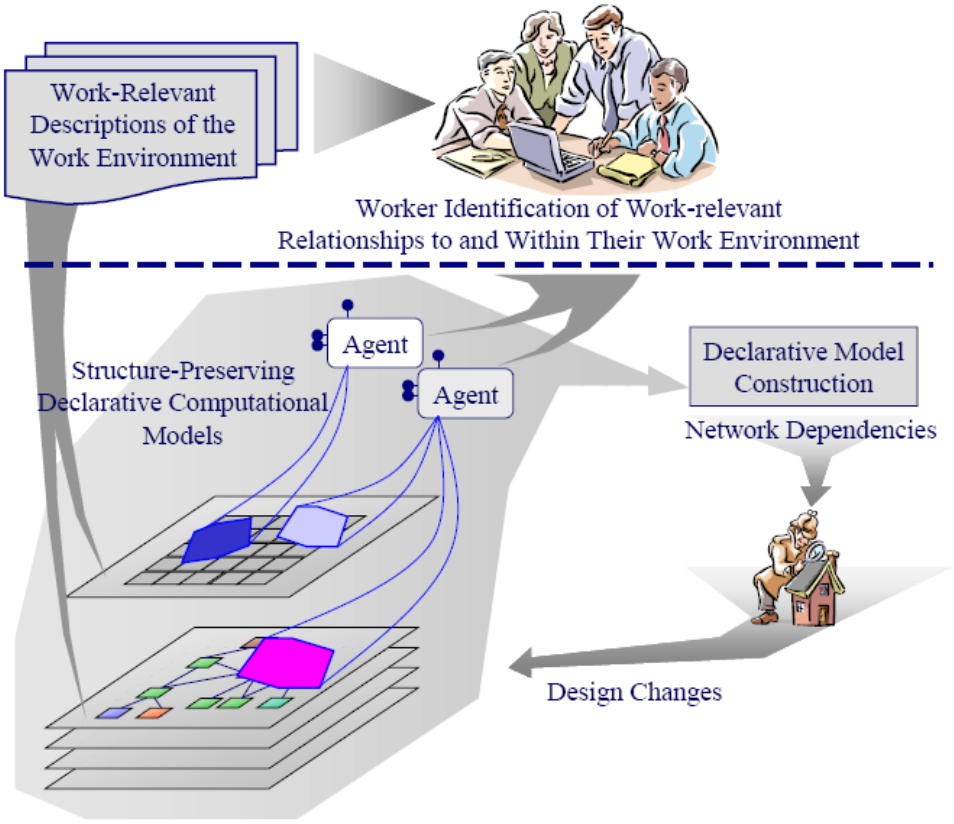

Figure 5: This thesis' approach to declarative modeling and querying for network analysis

The conceptual framework developed in this thesis enables modeling the socio-technical system and its elements computationally and in a structure-preserving manner. Through its model construction, simulation and query engines, it provides an ability to both explain and predict the impact of worker, component and network level transformations in socio-technical systems. As shown in Figure 4 and Figure 5, these analyses can be cyclic in nature, allowing for spiral evolution of design.

1.5 Overview of the Contributions

This thesis makes several contributions to the fields of cognitive engineering, agent-based modeling and simulation, and system design and analysis:

1. The primary contribution of this thesis is the provision of a conceptual framework to model and analyze transformations that are made at the level of workers, components and their networks. This framework is inherently structure-preserving and computational, thus enabling efficient analysis. This framework identifies:
    a. Which elements in the system and within the workers and components need to be modeled; and
    b. How these elements fit together to model the network of the work environment and the worker models.

    Furthermore, this framework is designed to enable operational and network analysis.
2. In identifying the relevant model elements, this thesis extends current cognitive engineering models, which typically only include the physical components of the work environment, by also modeling the non-physical components such as work-processes. These models allow for a more comprehensive view of socio-technical systems.
3. Unlike current cognitive engineering models that model only the means-ends and parts-whole relationships between the physical components, this thesis enables modeling the work environment through a variety of work-relevant relationships, again providing for a more comprehensive view of socio-technical systems.

4. This thesis provides for two kinds of analysis: 1) operational analysis that evaluates the impact on emergent system performance through simulations and 2) network analysis that identifies dependencies between system components through declarative specification and querying of their collective models and their interrelations.
5. This thesis constructs a domain-independent software architecture and simulation platform to computationally analyze the impact of transformations. This platform is a tool for practitioners and researchers in system design to test their intuitions about enacting system transformations through component, worker and network level changes. This architecture takes a structure-preserving and design-driven approach to constructing system models, thus significantly reducing the complexity of constructing and analyzing system models.

1.6 Research Tasks

This thesis describes the following four research tasks:

1. A review of relevant fields to build a foundation for the conceptual and practical developments of this research.
2. Development of work-relevant, structure-preserving constructs for modeling the work environment and the workers, as defined in a conceptual framework.
3. Development of a software architecture and a simulation platform for computational analysis of transformations in socio-technical systems.

4. Demonstration of these developments through a case study which used the framework to develop models of an air traffic control system, and which used the simulation platform to conduct network and operational analyses of this system.

1.7 Overview of the Thesis

In keeping with the research tasks, this thesis starts with a review, in Chapter 2, of existing approaches to modeling, analyzing and designing socio-technical systems. This review identifies the strengths and weaknesses of existing approaches for computational modeling of large-scale socio-technical systems. Literature in agent-based modeling is explored to draw out the core fundamentals for modeling an autonomous and interactive worker. The fields of agent-based simulation and system modeling architectures are explored to draw out the best practices for developing the software architecture and the simulation platform.

Chapter 3 establishes the core modeling constructs for the work environment and the worker, thus building the conceptual framework. Chapter 4 details a software architecture and simulation platform that enables analysis of transformations in a structure-preserving manner.

A case study in modeling, simulating and analyzing an air traffic control system is then used to demonstrate the usefulness of this work in Chapter 5. This socio-technical system is transformed with the use of work-processes in the work environment, thus giving the opportunity to demonstrate this work's ability to adequately model and analyze such transformations.

Finally, Chapter 6, after summarizing the theoretical contributions of the thesis, discusses further extensions of the work and its contributions to the design of socio-technical systems.

CHAPTER 2

LITERATURE REVIEW

This chapter discusses relevant concepts from the fields of Cognitive Engineering, the models of which form the basis of this work, Agent-based Modeling and Simulation, the principles of which are used extensively, and Modeling for System Design since system designers are the intended beneficiaries of this research.

2.1 Cognitive Engineering

This section reviews the models of human work from the field of Cognitive Engineering. Since this thesis focuses on analyzing socio-technical systems where the dynamics of interest are human work and the emergence of system performance from the collective work of the workers, a review of the theoretical underpinnings of this field provides several insights for developing this thesis' conceptual framework.

Cognitive Engineering, also referred to as Cognitive Systems Engineering, is a field of scientific study and engineering practice that emphasizes human centered design and analysis. Its scientific endeavor has historically been directed towards modeling work for the purpose of designing interactive systems, i.e., systems with which workers interact to accomplish their work.

Even though the term Cognitive Engineering was coined in 1981 by Norman (Norman, 1981) and most of the conceptual foundations for Cognitive Engineering were laid down

in the 1980's, Rasmussen had long been working on the engineering foundations of the field (Rasmussen, 1976). Cognitive Engineering primarily evolved to support worker operations in the face of demanding and uncertain situations and to ensure safety in critical operations such as nuclear power plants and emergency health systems (Rasmussen, 1976; Rasmussen, 1988; Vicente, 1999). Therefore, primary applications of Cognitive Engineering have been in deriving work practices or procedures that afford safe, efficient and robust performance, and design of technological aids and decision support systems (information systems) for critical operations.

Since its inception, the field of Cognitive Engineering has followed a logical progression. In the 1980's, researchers concentrated on establishing and defining the constructs that characterize work and then utilizing these constructs to model work. This was followed with development of methods for eliciting the knowledge that populate those models and methods that employed the models towards design of 'systems' such as technological aids for individuals or small teams. These methods represent the current core of engineering practices in Cognitive Engineering. The principles of cognitive engineering are also starting to be combined with the principles of systems engineering to derive methods, tools and practices that support the engineering of large-scale systems such as military organization command and control systems (Eggleston, 2002).

So far, the Cognitive Engineering community has concentrated on design of 'interactive systems' used by human workers to perform their operations, with particular attention to safety and time critical operations. Such systems include technological aids, information systems, control panels, and decision support systems. Vicente, in his book on Cognitive Work Analysis (CWA) (Vicente, 1999), increased the scope of the field to include the

'complete socio-technical system', i.e., the complete work environment, the tasks there in and the distribution of information and worker competencies. However, in practice, CWA has primarily been applied for designing interactive systems.

More recently there has been a shift in the use of the principles and knowledge base of Cognitive Engineering with respect to the kind of 'systems' that are the subject of design. Shah and Pritchett (Shah and Pritchett, 2005) have discussed using multiple dimensions of design factors in the work-domain for designing the broader socio-technical system; that is, instead of concentrating on design of technological aids, researchers are now also concentrating on changing the features and constraints of the work environment so that work in the system is safer and more productive. Cognitive Engineering principles have also been applied for determining information distribution in team activities (Sperling, 2005). In keeping with this newer perspective, the term 'system' is used in this thesis to refer to the whole system, including all its aspects that affect the workers, as the subject of design.

This review develops insights that are relevant to this latter definition of the system. The following discussion first introduces some conceptual approaches to modeling work and methods that employ those concepts, noting throughout the insights they provide about modeling the work environment in relevance to human work. Limitations of current frameworks are identified thus bringing to light the requirements for expanded models of the work environment.

2.1.1 Modeling Work

Merriam-Webster Dictionary defines work as a physical or mental effort or activity directed toward the production of or accomplishment of some objective or result (Merriam-Webster, 2003). Work is always purpose oriented, i.e., it is targeted to accomplish a goal; it does not have to be successful to fit within the definition of work. Furthermore, work does not have to be a physical activity; it can be cognitive instead. Cognitive Engineering was born to primarily support the latter.

Moving to the more technical definition in Cognitive Engineering, work is characterized in two ways. Most early models of work considered it as *problem solving*. This view assumes that the work to be done is a problem that the human is trying to solve, and that this problem is the driving goal of the work itself. Work is viewed in terms of tasks and task structures and modeled as event driven processes by which cognitive goals could be accomplished by the human. This "cognitivist" view models work in terms of functions and cognitive constraints of the worker (Eggleston, 2002; Vicente, 1999).

The second view defines work as *problem solving in context*. This view characterizes work as problem solving efforts arising not only due to the goals, but also in response to the work environment. The work environment can change during the performance of work activities, requiring the workers to adapt. Thus work is viewed in terms of the relationship between the work environment and worker's activities. This is the environment centric or "ecological" model of work (Eggleston, 2002; Vicente, 1999). Proponents of the ecological view claim that they are more robust through consideration of situated cognition, i.e., cognition that is adaptive to changes in the environment,

including unexpected events outside the task structures anticipated by the cognitivist view (Eggleston, 2002; Vicente, 1990; Vicente, 1999).

A distinction between the two views lies in their underlying models of work and their relationship to the work environment. Cognitivist approaches such as Cognitive Task Analysis (Schraagen, Chipman et al., 2000), Hierarchical Task Analysis (Shepherd, 1998) and other forms of behavioral analysis also analyze the work environment when developing models of work, but they employ task based models to express it in terms of functions to be achieved; elements of the environment are only captured within the models of the tasks. Thus, changes to the work environment are reflected indirectly by rebuilding existing and new models of work tasks and task structures. In contrast, ecological approaches such as Cognitive Work Analysis (Eggleston, 2002; Rasmussen, Pejtersen et al., 1994; Vicente, 1999) and Cognitive Triad (Roth, Patterson et al., 2001; Woods and Roth, 1988) maintain an explicit representation of the work environment from which they derive the models of worker tasks; thus, changes in work environment can be modeled directly, and models of work derived from them.

In essence, ecological approaches concentrate on the structure of the work environment and representing it in relation to the work, while cognitivist approaches concentrate on representing the tasks and cognitive constraints of the worker. For effective evaluation of a transformation's impact on worker and system behavior, a constructive combination of these two approaches is required. Of particular importance to this combination are the relationships between the work environment and worker activities as examined in the next section.

2.1.2 Modeling the Work Environment

This section discusses existing approaches to modeling the work environment in relevance to the work and to the workers. Furthermore, this section discusses how those models have been used for work analysis and for designing interactive systems. This sections draws valuable insights about the general characteristics of these models that may be used to develop a more comprehensive model of the work environment.

Ecological approaches define the work environment as including all objects and technologies in the system independent of any particular worker, task or goal (Vicente, 1999). Worker behavior is regarded as being shaped by the work environment. The work environment is seen as having intrinsic features that afford and constrain the workers towards achieving their goals. Ecological approaches view work as activities of the workers emerging from the goals they want to achieve and the affordances and constraints of the work environment. Ecological approaches posit that, if guidance about the goal state, affordances and constraints is provided, workers have freedom to choose or evolve task strategies for achieving their goals. Such an approach, first, enables workers to employ their creativity and devise task strategies that may be more efficient than those prescribed normatively, and second, enables them to be robust in the face of unexpected situations.

The worker is seen as an adaptive problem solver who is capable of exploring the work environment and deriving strategies to achieve the goal. The worker chooses his or her actions based on his or her goals, the perceived situation and the knowledge of the work environment, i.e., cognition is considered as situated. This view therefore argues that to

appropriately model work both the work environment and the worker approach to using the work environment must be represented.

Cognitive Work Analysis (CWA) (Rasmussen, Pejtersen et al., 1994; Vicente, 1999), one ecological approach, models the physical objects in the work environment on a two-dimensional representation known as the Abstraction Decomposition Space (ADS). The work environment is specified in terms of functional hierarchies, i.e., relationships of what in the work environment can help achieve which goal, and parts-whole relationships, i.e., which objects make up a sub-system and which sub-systems make up the system. Using this specification, one can identify which physical components would be used to achieve a specific goal, and the chain of sub-goals and sub-components needed to achieve a given goal.

Such a model of the work environment can be used by the worker through a generic problem solving approach such as the Decision Ladder (Rasmussen, Pejtersen et al., 1994; Vicente, 1999) to derive and rudimentarily define which tasks need to be done to accomplish a given goal in the given work environment, independent of who does it and how they are done. Essentially, it generates an input-output mapping for a given goal and the given physical structure of the work environment. Further application of another problem solving approach called Information Flow Maps helps the worker to explore possible strategies from a set of available strategies in deciding how the task may be achieved. Thus, through knowledge of those aspects of the physical world as shown in an ADS and through application of some general problem solving mechanisms, a worker can derive the tasks that accomplish work.

Such an approach to modeling and using the work environment assumes that the whole work environment has been modeled using the two dimensions on the ADS, and the appropriate parts of this knowledge are available to the worker to explore all possibilities using the problem solving mechanisms. Thus this approach is essentially meant to support the workers in doing their work by identifying the knowledge to the worker through training and displays, and through the assumption that the worker knows how to use the knowledge. CWA models have primarily been used to solve fault diagnostic problems, or to design distribution of information in physical systems.

Another ecological approach to modeling the work environment and using it towards work analysis is based in the work of Woods (Potter, Elm et al., 2002; Woods and Roth, 1988). Apart from the fact that this approach stresses the importance of the semantic completeness of external representation of knowledge to the worker as much as the work-relevant relationships amongst components in the environment, this approach also categorizes the work environment as comprising of different components which include decision problems and information elements. This approach models the decision problems that the workers are faced with in a hierarchy (or a tree) using means-ends relations. Each decision problem is associated with the information in the work environment that is needed to make a decision with respect to the decision problem. A worker, when faced with a decision problem, uses this model to traverse through the hierarchy to, first, find the chain of required sub-decision problems, and, second, to find the information in the work environment needed to solve the problem. This model is also used to support workers, but for decision-making, a very specific kind of work.

Looking at these two models, certain common characteristics can be drawn out. First, the work environment is viewed as including 'things' in the system that affect the choice of worker activities or affect the outcome of the work, irrespective of which worker is in the environment, what their goals are and which tasks may use them. Second, these 'things' can be physical or may include some non-tangible elements such as information. For example, (Ockermann and Pritchett, 2000) model procedures as a part of the system environment. Third, there are work-relevant relationships between these things, the knowledge of which can be used, together with other internal or external knowledge, to derive worker tasks. Fourth, the existence of means-ends relationship between these 'things' and the goals of the workers is important for the 'things' to be relevant to work. Fifth, depending on which kind of 'thing' is modeled in the work environment and which relationships are modeled between these 'things', the models support different kinds of work; for example, means-ends relations and parts-whole relations between physical objects support fault diagnosis, but decision-problems and information on means-ends relations may support decision-making.

There are a few limitations too. First, these models have evolved primarily for design of interactive systems that support humans in their work. They also help derive task sequences relevant to the structure and the state of the work environment. However, it still needs to be seen how they can be used to analyze transformations in a socio-technical system. Second, whereas means-ends relations are necessary to identify things relevant to work and parts-whole relations can support a variety of tasks, a number of other relationships within the work-environment maybe important for different kinds of work. Third, though the current approaches collectively represent both physical and non-

physical components of the work-environment, they have not yet been fully integrated to leverage their combined strength in comprehensively analyzing socio-technical systems. Furthermore, the pool of different types of components that they collectively represent is not comprehensive and should be expanded for analysis of worker, component, and network level transformations in large-scale socio-technical systems.

2.1.3 Analyzing Systems

This section discusses how the interactive systems that are designed using cognitive engineering are evaluated, and what insights can be taken from these approaches to evaluation in developing a transformation analysis approach. To be useful for the purpose of design, any modeling and analysis framework has to enable evaluation and comparison of design alternatives. Rasmussen (Rasmussen, Pejtersen et al., 1994) lists general methods of evaluation that could be used for analyzing interactive systems produced by cognitive engineering. These methods are based on evaluation categories termed as *Analytical* and *Empirical* in (Rouse, 1984; Rouse, Frey et al., 1984).

Empirical methods establish a controlled experimental work situation creating a defined constraint envelope around the subject and study whether the subjects' behavior in this envelope satisfies design objectives. Rasmussen characterizes these methods as best suited for separate tasks or functions for which a reasonable level of operational skill can be developed and appropriate performance criteria found. Rasmussen also notes that empirical methods are not suited well for systems/tasks with a number of uncontrollable variables. These methods have primarily been employed to evaluate issues related to human capabilities in interacting with the designed systems.

Analytic evaluation, on the other hand, emphasizes the functional rightness of the system. In this approach, issues related to the contents of the information provided by the system are evaluated analytically for satisfaction of work requirements, while issues related to its form or presentation to humans are left out to be evaluated empirically. Thus this approach evaluates the system for effectiveness, i.e., its ability to get the work done in principle, before human ability or task efficiency is evaluated. Analytic approaches are therefore recommended for evaluating systemic changes or epistemic changes (for definitions of these changes the reader is referred to (Benda and Sanderson, 1998)).

Empirical methods can be viewed as performing an operational analysis with the use of controlled experiments and longitudinal studies on prototypical or real systems. They combine the interactive system and the worker in a controlled environment and study the operational characteristics of the combination, particularly the impact of the combination on human behavior if the impact of human limitations on the performance of their interactive work. However, to date such analysis has been limited to evaluation of small teams and interactive systems, especially when relying on human-in-the-loop tests.

Analytical methods such as that used by CWA are capable of performing network analysis. For example, ADS can be used to evaluate dependencies between functional objectives and the physical resources in the system. This dependency can be extrapolated to identify dependencies between information required for a particular function and the physical source of that information. Such a dependency analysis (network analysis) can be used to evaluate whether the current system configuration, in terms of which resources have been made available for which functions, is conducive to the possibility of accomplishing work, irrespective of whether humans are capable of using that

information or not. However, since analytical approaches such as CWA use descriptive models, they are currently limited in the complexity and size of networks they can analyze and the level of detail they can account for in their models of the work environment and worker behavior.

A use of analytic approach for design evaluation can be seen in (Benda and Sanderson, 1998). This work enhanced the abstraction decomposition space to create a model of the work domain with the design intervention accommodated in the model. The model was then analyzed, using their personal judgment, to predict the impact of design interventions. Even though an expert, using this approach, may provide an accurate prediction and a reasonable recommendation, this method is impracticable for large and complex systems, where there are too many variables and interactions to be practically accounted.

In summary, in their current state, empirical analysis is largely limited to analyzing workers and their interactions while analytical analysis primarily emphasizes on the work environment. While these analytical and empirical methods could be used sequentially to perform complete system analysis, they have not yet been integrated for simultaneous analysis of the network and operational issues. As a result, evaluation of systems designed using cognitive engineering has mostly been limited to human-in-the-loop studies or through use of prototypes of the interactive systems (for example, (Potter, Elm et al., 2002)).

Computational techniques such as simulation are necessary for evaluating large-scale systems with sufficient detail. Moreover, personal judgment and lack of formal analysis through concrete constructs makes the analysis less credible and less repeatable. A

systematic process with concrete constructs that allow for formal analysis helps reduce subjectivity and contributes to the quality of the analysis. Formalization or specification of semantically concrete constructs also makes it possible to represent the models in a computational form, thus making it practicable to analyze large-scale systems with the use of computers. Whereas there are computational architectures that evaluate 'system' performance, these are largely limited to examining interactive systems. That is, there are computational models of human performance or of task performance that describe human interactive systems. Computational architectures for system design evaluation exist primarily for the cognitivist approach to evaluate efficiency of specific task models targeted at specific driving goals in closed or well-defined environments. Computational approaches such as that used by AirMIDAS (Corker, 1994) analyze adaptive behavior of humans in rich and open environments but their purpose to date has been analysis of human performance and human interaction with technology, rather than network level transformations in the system or their operational impact.

For the ecological approaches, there are drawing, editing and tracking tools for producing modifying and storing the ecological models, such as the CACSE (Computer Aided Cognitive Systems Engineering) for FAH (Functional Abstraction Hierarchy) models and WDAW (Work Domain Analysis Workbench) for the ADS models in CWA (Eggleston, 2002; Skilton, Cameron et al., 1998). However, these tools do not provide any assistance in designing and evaluating design alternatives. Rather, they are used essentially for requirements definition of complex mission systems or for knowledge representation for decision support (Eggleston, 2002; Potter, Elm et al., 2002).

Thus, these approaches have largely focused on human-interactive systems and would not scale to larger systems. In addition, since the performance of a socio-technical system emerges from factors both in the work environment and the worker, evaluation of design alternatives for socio-technical system should consider both the behavior and limitations of the workers and their work environment, and the overall functional rightness of the system. Thus, approaches to evaluating socio-technical systems should combine the empirical and analytical approaches, and the underlying models should be specified through concrete computational constructs.

2.1.4 Summary and Discussion

This section discussed two cognitive engineering viewpoints, the cognitivist and the ecological. Ecological approaches consider the work environment as affording and constraining the worker in achieving their objectives. Knowledge about the affordances and constraints in the work environment can be employed by an adaptable (generic problem solver) worker to perform robustly in the face of unexpected situations that may not have been accounted for in preconceived models of tasks. Thus, the ecological models' ability to explicitly represent the work environment outside the models of tasks, yet in relation to work, is useful for analyzing transformations to the work environment. However, for complete understanding of system performance, internal characteristics of workers and their internal mechanisms in relation to work, as examined by cognitivist approaches, should also be modeled. An integration of these two approaches is therefore required.

The relationships between system components modeled in the work environment were related to the nature of the work they support. Work may be categorized based on the class of problems that the worker has to deal with. For example, diagnosing a system for faulty components is one class of work problems, while decomposing an overarching decision problem into smaller more tangible decision problems is another class of work problems. Each approach to modeling and analyzing systems is intrinsically more suited to one or more of these natures by virtue of what is modeled and the set of relationships between the modeled elements.

Existing ecological approaches map well to certain specific natures of work but not to others. CWA models the structural elements over the means-ends and parts-whole relationships, which may be best suited for work problems such as diagnosing faulty components in physical systems and discovering structural resources. Potter et al's approach models decision problems over the means-ends relationships; it is therefore best suited for decomposing larger decision problems into a combination of more tangible decision problems.

While the existing approaches collectively cover a wide set of classes of work problems, they are not suited for all problems such as event driven selection of work-processes. A number of other environmental components and a number of other relationships may need explicit modeling to cover those classes of problems. Additionally, these approaches have not been integrated to leverage their collective strength in representing and analyzing the work environment. Hence, an approach is required capable of aggregating and integrating the relationships modeled by the existing approaches and also incorporating previously not modeled relations.

As noted, few computational constructs to enable objective and credible analysis that informs design of large-scale socio-technical systems in contrast to interactive systems and small teams. Current approaches to evaluating systems have concentrated on distinct parts of the system, i.e., only humans or only the work environment, but have not been integrated to analyze complete socio-technical systems for transformation at the component, worker and network levels. Development of concrete computational constructs for such analysis remains to be an area of research.

2.2 Agent-Based Modeling and Simulation

Recent developments in software engineering, artificial intelligence, complex systems, and simulation science have placed an increasing emphasis on the concept of agents (Bargiela, 2000; Jennings and Wooldridge, 2000; Parunak, 2000; Parunak, Savit et al., 1998; Russell and Norvig, 1995; Weiss, 2000). The term *agent* has been used to mean anything from a mere subroutine or object to an adaptive, autonomous, intelligent entity (Franklin and Graesser, 1996; Hayes, 1999; Wooldridge, 2000). This thesis uses Hayes' definition of an agent as an entity with (1) autonomy, i.e., the capability to carry out some set of local operations and (2) interactivity, i.e., the need and ability to interact with other agents to accomplish its own tasks and goals (Hayes, 1999).

Autonomy dictates a few essentials. To behave autonomously, the agent has to proactively work towards some internally held goal(s). Thus, an agent has to be purpose or goal driven, irrespective of how or where these goals are generated and how they are transformed to actions that achieve the goals. Additionally, in order to achieve its goals, the agent will seek to change the state of the environment. Thus, an agent should be able

to interact with its work environment, both by sensing it and by acting upon it (Figure 6). How it chooses the actions, and how and where it stores the semantic association between the sensed signals, the goals and the actions, is not specified in this definition.

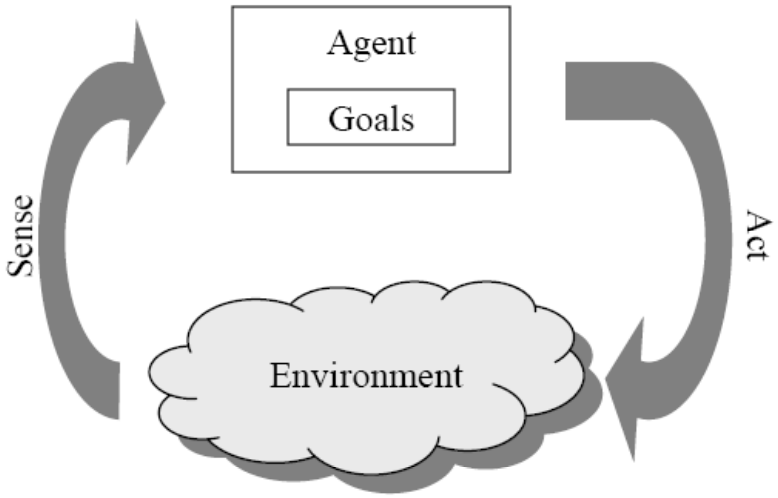

Figure 6: Notion of an *Agent*

The second tenet is interactivity. A single agent only works with its work environment, but when there are more than one agent in the system, they interact. They might interact passively, due to the changes in the work world observed by one as a result of the actions of the other (Holland and Melhuish, 1999), or intentionally through the use of direct communication, coordination or collaboration (Durfee, 2000; Huhns and Stephens, 2000; Singh, Rao et al., 2000). For all work that involves more than one agent in the same work environment, this aspect of the agent paradigm needs to be addressed.

When using an agent-based paradigm to model a domain, the analyst uses the abstraction of an agent as one of the basic units of analysis. The process of abstraction then might delve deeper into what constitutes an agent or their interactive mechanisms. However, the modularization of an agent follows the natural boundaries among individuals. The migration from a simulation model to the real world and converse is therefore much more natural in agent-based modeling compared to a number of other modeling paradigms. This structure-preserving nature of agent-based modeling is therefore more suitable for designing systems where design intervention is made through local variation in individuals or other entities in the system. Furthermore, due to its structure-preserving nature, agent-based modeling is most appropriate for modeling and analyzing domains characterized by a high degree of localization and distribution and dominated by discrete decisions (Parunak, Savit et al., 1998).

The agent paradigm has found wide application in a variety of domains ranging from modeling and simulation to applications that autonomously or semi-autonomously act on behalf of their users in making decisions or performing specific activities. Common to all these applications is the notion of problem-solving and performing tasks in accomplishment of some goals. Building on the definition of work and a worker from §2.1.1, an agent can be viewed as doing work, thus itself being characterized as a worker. Such a characterization, accompanied with the other traits outlined in this section, makes an agent-based model a natural choice for modeling a worker.

2.2.1 Agent Architectures

Once the analysts have decided on ABM as the paradigm to abstract the problem domain, the next step is to define the constructs in more concrete ways so that they can be used in modeling and analysis. These constructs are used to formulate the agents, their interactions and their work environment. This task is mostly theory development and is targeted at the specific types or categories of problems that the theory addresses. Several theories have been developed in agent modeling and creation. An overview of formal theories can be obtained from (Singh, Rao et al., 2000); note this thesis develops its own conceptual framework in Chapter 3 and modeling architecture in Chapter 4.

Theory development is followed by principled construction of the abstracted entities. One such entity in the system model is, of course, an agent. Whereas the exact constituents of an agent vary with the constructs used in a theory, over the years researchers have come up with classes of agent architectures that can be used to address a range of problems. In the following, general characteristics of agent architectures are discussed. Since the model of socio-technical system in this thesis employs models of agents, this discussion will be helpful in understanding the modeling architecture developed in this thesis.

As discussed in the previous section, an agent is an autonomous and interactive problem solving entity, and ABMS is primarily used to analyze systems through emulation. This premise has led to the creation of two primary strands of agent architectures. The first strand is Artificial Intelligence (AI) and robotics and the second strand is Cognitive Modeling (CM) and Human Performance Modeling (HPM).

AI based architectures aim at emulating intelligence while disregarding structural or functional conformance to the internals of the entity whose behavior is being emulated. For example, if the aim were to emulate a bird, AI would come up with solutions like an airplane, a helicopter or a hovercraft. These solutions are very different in their structure and their internal functioning from the motivating object, i.e., a bird. However, they still achieve the function of flying and on some dimensions to a much better level of effectiveness and efficiency. The success of an AI agent is judged by an agent's ability to emulate intelligence as measured by an *external* observer (Brooks, 1991; Turing, 1950).

On the other hand CM and HPM aim at not only emulating the behavior (as observed by an external observer), but also on emulating some aspects of the internal structure and functioning of the object being emulated, which is the human cognition or the human perceptual-cognitive-motor capability. Thus the CM/HPM strand, if aiming at developing a flying object, when the motivation is being drawn from a bird, would try to emulate the birds body, its wings and its flapping and gliding to achieve the function of flying.

2.2.1.1 Cognitive Models and Human Performance Models

Research on cognitive models and human performance models was motivated by the thought that software agents that perform tasks in the same way as humans could be used to evaluate proposed system designs without the need to conduct these types of evaluations with actual workers. Such models can provide a priori performance predictions of how well a certain system will support workers by assessing factors such as how easy the system will be to learn and use, the workload it imposes, and

susceptibility to errors. However, they represent highly controlled and directed behavior on part of the human being modeled, and a number of other contributing factors (such as emotional state of the worker) are unaccounted for. Regardless, these models provide good predictions for the specific controlled tasks that they model. These models have primarily been employed for steering human in the loop studies for critical operations using novel systems.

In (Yoshikawa, 2003) the necessary components of a computational human model are discussed. The three elements are the knowledge model, the process model and the control model. The knowledge model relates to semantic content, its storage and retrieval. Generally, the knowledge model consists of a representation of the state of the work environment and the internal state of the agent, the goals of the agent and the cause-consequence relationships. In other words, the knowledge model represents the beliefs and desires of an agent. The process model consists of modules that process the signals in the work world (perceptual modules), that process the knowledge stored in the human memory to derive action plans that need to be executed to achieve the agents' objectives (cognitive modules), and that actually execute the chosen actions (executive modules). Each of these modules is associated with elementary cognitive processes that can be combined into a larger process or control model that exhibit the agent behaviors in response to the work-conditions faced by the agent. Control models are primarily understood as those models which decide on the order of executing a sequence of elementary cognitive processes into a whole cognitive system interaction of the human. However, there is more to control models than just information processing through integration of modules; information processing mechanism may be chosen with respect to

the problem at hand and the circumstantial problem constraints. For example, a reflexive response may be chosen in case of time constraints and a more deliberative process may be chosen in the case of availability of time.

In further defining these models and theories, the research communities in human computer interaction (HCI) and cognitive psychology have come to a consensus on the basic elements of these models. For example, the knowledge model is purported to consist of the long-term memory and short-term or working memory. Working memory is used continuously by the decision-making process and problem solving activities and is much faster but much smaller compared to the long-term memory. The long-term memory holds all the human's knowledge. Similarly, the perceptual, cognitive and executive modules are considered to be the core modules in the process model. There are distinctions between theories in how the same elementary cognitive processes in these modules are modeled and how they interact with the knowledge models; regardless, these are the core components of every process model. Information processing models employ different protocols of interaction amongst these modules and their associated processes to compose the overarching cognitive process of a human. Similarly, there is a consensus on existence of the four contextual control modes: scrambled, opportunistic, tactical and strategic (Hollnagel, 1993) that govern the choice of activity (Rasmussen, 1983; Rasmussen, Pejtersen et al., 1994).

Theories in cognition have conceived a number of constructs that identify with each of these models. For example, knowledge can be of four types: Declarative, procedural, episodic and iconic. Knowledge represented in one of these forms can be utilized by an information-processing model to draw inferences that suit specific functions. For

example, semantic associations are best represented as declarative knowledge, while procedural knowledge is best suited to represent skills. Since the knowledge model is very closely related to a process that will use this knowledge for accomplishing work, knowledge representation and the selection of data structures in this model is very closely related to the choice of information processing model or the planning module in the process model.

Similarly, there are theories that represent elementary cognitive processes. For example, signal detection theory is employed to model the elementary cognitive processes of attention (Treisman and Gelade, 1980; Zandt, Colonius et al., 2000), and elementary cognitive processes may be represented as mechanisms for the choice of productions, i.e., knowledge related to elementary actions (Anderson and Lebiere, 1998; Newell, 1990).

Software agents that model humans aggregate a selection of these theories and associated models that address specific functions in human performance and operation and integrate them as modules in software agents. For example the Model Human Processor (MHP) and the EPIC architecture are the very basic human models that are equipped with the basic process modules that interact with the task environment and the internal knowledge store that is also available in the human architecture (Byrne, 2003; Yoshikawa, 2003). The concept of environment is a state space that relates to work environment and the tasks at hand.

The interaction between these modules is based on some assumed information processing architecture that governs the structural, temporal and information exchange relationships amongst these modules. MHP is used by GOMS, which employs an overarching information processing model that transforms goals to operators that use the elementary

cognitive processes. The control model is essentially a reduction mechanism that can select the operators by progressively breaking the higher goal into sub-goals (John and Kieras, 1994). The EPIC architecture is primarily used by the EPIC processor. The knowledge is represented as production rules and the cognitive process selects the productions that should be employed to do the complete information processing (Kieras and Meyer, 1996). AirMIDAS has a similar human operator model, although the knowledge is not represented as productions but rather through activity maps, daemons and updatable world representations (Smith and Tyler, 1997). AirMIDAS is also more advanced than the previous two in that it has an explicit representation of contextual control modes in its control model (Verma and Corker, 2001). Similarly, the skills, rules and knowledge paradigm proposed by Rasmussen (Rasmussen, 1983) can also be used with the decision ladder (Rasmussen, Pejtersen et al., 1994; Vicente, 1999) to generate a problem-solving (agent) architecture where the knowledge model is represented as an Abstraction Decomposition Space (Rasmussen, Pejtersen et al., 1994; Vicente, 1999) and some parts of the process and control models are represented as a decision ladder; however, some aspects of behavior such as temporal dynamics would need to be specified by some other model. A concrete implementation of such an agent architecture is not known to the author at the time of this writing. Other similar architectures include Soar (Lehman, Laird et al., 1993) and ACT-R/PM (Anderson, 1993; Byrne, 2001).

2.2.1.2 AI Architectures

Over the last few years, a number of AI agent architectures have been proposed, ranging from simple reactive agents to agents with complex cognitive processes and architectures (Russell and Norvig, 1995; Weiss, 2000; Wooldridge and Jennings, 1995; Wray, Chong

et al., 1994). (Wooldridge, 2000) discussed four general classes of AI agent architectures:

1. *Logic based agents,* in which decision-making is realized through logical deduction. This is the traditional approach to artificial intelligence where the world is represented symbolically through the use of logical formulae. The agent transforms between inputs and outputs through the use of logical deduction or theorem proving. A summary of logical formalisms used in agent architectures can be found in (Singh, Rao et al., 2000). These architectures were primarily devised for game playing agents and problem spaces where the complete problem space could be represented through the use of concrete logical constructs. These architectures were primarily meant to encapsulate autonomy in logical problem solving.
2. *Reactive agents,* in which decision-making is implemented in some form of direct mapping from situation to action. Reactive architectures were primarily devised for robots working under time pressure. These architectures work through the use of simple rules representing reflex actions (Arkin, 1998; Brooks, 1986; Brooks, 1991). Again, these architectures are primarily meant to encapsulate autonomy through use of simple behaviors such that the agent is robust to time constrained situations.
3. *Belief-Desire-Intention (BDI) agents*, in which decision-making depends upon the manipulation of data structures representing the beliefs, desires, and intentions of the agent (Bratman, 1987; Bratman, Israel et al., 1988; Rao and Georgeff, 1991; Singh, Rao et al., 2000). This architecture has its roots into the philosophical

tradition of practical reasoning. Beliefs, i.e., the world model of the agent, form the agent's knowledge base and are usually represented through modal logic. Desires are usually represented as conditions that the agent wants to achieve. Desires can be inconsistent, and the agent may not be able to achieve its desires. Goals are the subset of desires that can be achieved by an agent. Intentions are usually understood as the states that lie on an agent's plan of action toward its desires. An agent achieves its desires through the formulation of intentions and the application of effort to achieve those intentions. This architecture encapsulates autonomy and models proactive behavior through abstract but practical thinking.

4. *Layered architectures,* in which decision-making is realized via various software layers, each of which is more-or-less explicitly reasoning about the environment at different levels of abstraction. In such architectures, there is usually a distinction between the functional abstraction of the tasks handled by each layer. On the minimum, there are two layers: one handling the reactive or reflex behaviors of the agent, and the other handling the more proactive or cogitative functions. Two types of layering exist: horizontal and vertical. In horizontal layering, each layer operates in parallel with the others in generating options or choices for action; in vertical layering only one layer acts at a time depending on the task and the sensory input. For examples of layered architectures one can refer to (Brooks, 1986; Brooks, 1991; Ferguson, 1992; Muller, Pischel et al., 1995). These architectures are more of a hybrid of reactive and deliberative architectures where there are separate parallel modules that operate on both levels

(reactive and proactive) and control passes along between the these modules to achieve context-adjustable intelligence (similar to contextual-control modes discussed in CM and HPM architectures discussed in the previous section).

Apart from these categorizations, there have been a number of dedicated architectures which have been used for tackling specific problems. Those architectures have not been discussed here in the interest of space and relevance.

2.2.1.3 Discussion

In essence, any cognitive architecture, based on AI or human cognition, represents the world symbolically and stores this information in some kind of data structures. A process is applied to pull out the relevant information with respect to the stimulus in the work environment and the current goal. A problem solving transformation on this information is then applied towards defining the tasks, the execution of which is expected to lead to the achievement of the goal. In some cases the stored information is the transformation itself, i.e., the retrieved information can be directly applied by the processor for generating the action. Examples of such architectures include those that employ if-then rules, production systems and artificial neural networks.

In the case of AI, the storage structures may be trees, artificial neural networks, Bayesian networks, beliefs represented as modal logic, etc. The search usually involves some brute search algorithms such as depth-first search, or informed search methods such as the A* (Yokoo and Ishida, 2000). In case of cognitive models, storage structures represent human cognitive elements such as the working memory and the long-term memory. The information may be stored as semantic networks or through some other cognitive data-

structures. The search process may involve methods such as associative discovery or semantic activation.

Even though the 'cognitive' aspect (i.e., decision-making) has been the center of attention for most research, a number of researchers have concentrated on integrated architectures that also accommodate perceptual and motor faculties to complete the agent model. This is specifically true of human performance models, which are interested in evaluating performance constrained by human limitations. In case of AI agents, this usually hasn't been the issue except, of course, for robotic systems. In these architectures, specific attention is paid to the perceptual and executive modules and their interaction with the cognitive modules. In some cases these modules fit into a linear information-processing model where information flows sequentially from the perceptual module to the cognitive module and then to the executive faculties of the agent. A number of other architectures have more complex associations between these modules that enable the agent models to manifest expert behaviors (decision ladder in (Rasmussen, Pejtersen et al., 1994; Vicente, 1999)) different contextual control modes (Hollnagel, 1993; Rasmussen, 1983; Verma and Corker, 2001), or parallel processing for reflex actions and the more abstract thinking (Brooks, 1986; Muller, Pischel et al., 1995).

In case of linear information processing architectures, perceptual and motor faculties do not have to be modeled as separate modules and can be combined or fused into a single process module. The distinction is merely for the sake of representation and understanding rather than implementation or complex process composition. On the other hand, the other architectures represent separate modules that interact and integrate with each other to create a complete agent. Complex process architectures employing parallel

processing and interrupt or interlock mechanisms requiring process and temporal synchronization can be created with the use of distinct modules, thus representing phenomenon such as attention to alarms.

Perceptual and executive modules have a direct relationship with the environment or the agents' work environment (these relationships are discussed in the following section on agent environment). It is through these modules the agent perceives its work conditions and changes the work environment towards achievement of its goals, i.e., exhibits behavior. The performance of an agent or its intelligence is measured in terms of its behavior as perceived by an external observer. Agent accomplishes all work in the system through the use of these behaviors.

In general, the perceptual, cognitive and executive modules that envelope the agent architectures can each be considered as having three parts. One part is the processor that governs autonomous processing of each of these modules. The other part is the basic capability that is common to all agents of a general class. Finally, the third part is the skills that are acquired by individual agents and that complement their basic capabilities in their ability to do particular tasks or take particular actions.

2.2.2 Multi-agent Systems

The previous section addressed the structure of a single agent and the mechanisms that make it exhibit autonomy and intelligence. When creating an artificial agent for accomplishing some difficult tasks, one could think of designing an extremely capable and intelligent agent that would successfully achieve the objective. However, due to limitations of technology, there are a number of tasks that cannot be accomplished by a

single agent, but may be accomplished through synergism between a number of agents. In the real-world with human integrated systems this is done through establishment of organizations and formation of teams, and thus they have an organizational aspect helps humans accomplish functions beyond the capability of the individuals themselves.

Agents interact through either acting on their work environment (stigmergy (Holland and Melhuish, 1999)) or by acting on each other (communication). For example, an agent may change the state of the work environment by writing something on a black board, and thus interact with the other agent who is capable of reading the blackboard and writing back on it. Similarly, the agent may invoke a direct communication with the other agent where they will send a message that can be interpreted by the receiving agent through the use of some common semantics, i.e., communicate with the use of language and some communication medium.

Whatever the medium of interaction, multi-agent work can be achieved through effective coordination. In multi-agent community, coordination refers to use of protocols for performing interactive activities, i.e., through synchronization of interactive activities between agents (Huhns and Stephens, 2000; Singh, Rao et al., 2000). Researchers have identified several patterns of interaction that can be used to tackle specific coordination issues, for example (Hayden, Carrick et al., 1999). Whereas effective and efficient coordination mechanisms can help multi-agent systems achieve a number of organizational tasks, current models of coordination mechanisms assume both the agent roles and their organization are fixed.

Some multi-agent activities require that the agents dynamically organize to suit the tasks at hand. For example, organizations reorganize their human resources as per the demand

of projects that they get from their customers. For closed organizations, where each agent is fully aware of all other agents' capabilities and can utilize them as resources, centralized mechanisms can be used to determine coordination mechanisms, (Nair, Tambe et al., 2003; Nair, Tambe et al., 2003). However, when the reorganization needs to address the objectives of individuals, such as their liking of specific kinds of projects, centralized coordination mechanisms are not enough and individual's goal satisfaction has to be included. Various decentralized mechanisms such as negotiations, bargaining, auctions, market strategies etc. can be employed to achieve collaboration (Huhns and Stephens, 2000; Sandholm, 2000; Singh, Rao et al., 2000), i.e., satisfaction of individual goals while also achieving organizational goals.

In all the above multi-agent systems, the mechanisms that achieved multi-agent capabilities related to attributes of the agents, i.e., their goals and their behaviors. In these mechanisms, behavioral characteristics common to all agents are induced to achieve a system level social behavior (Castelfranchi, 2000). While these mechanisms for obtaining organizational attributes are understood as general theories of collaboration, these theories apply to intrinsic characteristics (intrinsic decision control algorithms) of the agents. In other words, social order is hardwired into the agents (Castelfranchi, 2000). However, most organizations live beyond the lives of the agents that constitute them. Adaptive and learning agents may take on new roles, and a common way of transforming organizations is to change their organization. As (Castelfranchi, 2000) points out, there is a need for continuous social control to maintain social order throughout the life of the organization. There has been some research where the organization-creating characteristics are seen as residing outside the agent and in the

organization or the work environment (Decker, 1994; Decker, 1998; Decker and Lesser, 1994). However, these works are mostly limited to task coordination strategies in multi-agent settings.

2.2.3 Modeling Agent Environments

An agent exists in an environment, i.e., what it senses and acts upon. Without the environment the notion of an agent is useless, primarily because the agent performs the work on the environment and its behavior is observed with respect to the environment. The early agent community viewed the environment as a state space consisting of observable and updatable state variables. Agents' behavior was contingent to this state space, especially with purely computational agents. However, (Odell, Parunak et al., 2002) point out a slightly enhanced view of the environment as an entity within which the agent exists, and which provides the medium of interaction among agents. They characterize the environment as having states as well as principles and processes that govern its behavior. Principles identify the environmental characteristics such as accessibility, determinism controllability, temporality etc. as pointed out in (Russell and Norvig, 1995; Wooldridge, 2000), and, going beyond the notion of physical environment they also hint on communication environment and the social environment that includes patterns of interaction or interaction protocols. However, this work (Odell, Parunak et al., 2002) does not develop any concrete theory or representing these aspects of the environment in a multi-agent system. In (Decker, 1994) relationships between tasks and executable methods were modeled as existing in the environment, but they were only

used for a priori design of coordination strategies and not as something dynamic that the agent would have to perceive and interact with while performing work.

Incidentally, most of the agent research has either been related to purely electronic agents or to purely physical agents (robotics). In the case of purely electronic agents, state spaces remain the appropriate model of the environment as the states are changed by processes exercised by the agents in the environment, thus giving it the dynamism. In the case of robotic agents there is no explicit need to model the environment as it already exists in reality. Researchers in robotics are more concerned with tackling the issues related to accessibility and uncertainty of the environment than modeling it as a separate entity in which the agents will exist, although, in the case of simulations of physical activities such as playing soccer, environment models are explicitly created to have state-space, principles and processes. There is a lot of work done on *virtual environments*, but that explicitly relates to designing multi-modal interfaces for humans rather than for agents.

2.2.4 Simulation Architectures

There are a number of general agent simulation packages available in the research community. These include Starlogo (StarLogo, 2004) and Netlogo (Wilensky, 1999) for simulation of natural social phenomena, Teambots and SoccerBots (TeamBots, 2000), and SimAgent (SimAgent) for robotic soccer simulation, XRaptor (Bruns, Mosinger et al., 2003) for simulation of continuous time physical robots and Swarm (Swarm). All of these general simulation packages are suited for agent behaviors characterized by continuous-time dynamics. Most of them are only suited for simple behaviors and very

simplistic decision-making and interaction. Moreover, these behaviors are modeled such that they cross the boundaries of modularity of agents and represent general characteristics of the modeled agents. They cannot be utilized for simulation of multi-agent systems that model complex socio-technical systems because they do not preserve the structure of socio-technical systems and are limited in the types of behavior they can represent.

Other simulation packages such as MIDAS (Smith and Tyler, 1997), GOMS (John and Kieras, 1994), Soar (Lehman, Laird et al., 1993), ACT-R/PM (Byrne, 2001), EPIC (Kieras and Meyer, 1996) and Team-Soar (Kang, 2001; Kang, Waisel et al., 1998) are suited for simulation of complex cognitive process in human or intelligent agents, specifically task performance in continuous time or based on discrete events. However, they are currently best suited for simulation and analysis of individual human-machine systems or small teams performing specific team tasks. Furthermore, these simulations fail to provide the appropriate mechanisms for disparate agents in the system, each requiring control of their own timing mechanism.

Besides these commonly known agent and multi-agent simulation platforms, there are a few proprietary but very advanced platforms that are used by small groups of researchers and practitioners. These include OpenCybele (IAI, 2004). This research has chosen to use the Re-configurable Flight Simulator (RFS) (Ippolito and Pritchett, 2000; Lee, 2002; Shah and Pritchett, 2005). RFS is well suited for multi-agent simulation of large-scale human integrated systems. Though its architecture and implementation was originally designed for simulation of aviation systems, it has an architecture allowing a wide range of agent behaviors and arbitrary number of agents. Of particular interest is its timing

mechanism that affords simulation of very different dynamics (discrete or continuous) in one simulation. Moreover, it provides control of temporal dynamics to each agent, while maintaining temporal synchronization between all agents in the simulation. Thus the timing mechanism is suited for modeling autonomy to a high degree of temporal fidelity. In addition, its object-oriented design allows for structure-preserving abstractions.

2.2.5 Summary and Discussion

An agent is an autonomous (proactive) and interactive entity, capable of controlling its decision-making and its own temporal dynamics. Several theories and internal architectures are available for principled construction of agents over varying degrees of intelligence. Similarly, several agent simulation architectures are also available.

The primary insight is that the agent paradigm analyzes the system with respect to behaviors, which is the basic unit of analysis in this thesis. It is specifically suited for modeling and analysis of systems that are marked by distributed and localized decision-making and are characterized by emergent behaviors. The most important characteristic of agent-based modeling is its ability to model these complex, localized yet global systems in a structure-preserving manner where the natural modularization of the models matches that of the modeled individuals.

As discussed in §2.1.4, to model and analyze a socio-technical system both the work environment and the workers should be modeled. The discussion in this section addressed the state-of-the-art for the latter, while the previous section discussed the former. Building up on the work in these two areas, a modeling and simulation framework for analyzing system transformations can be created.

2.3 Modeling Approaches for Supporting Design

To analyze the impact of system transformations requires understanding of how systems are designed in general so that the modeling and analysis approach can best aid the design process. Any system design evolves iteratively, continually cycling through design, implementation and evaluation as guided by the objectives of the stakeholders and the exogenous factors (Figure 7). All engineering disciplines recognize that modifications to the system cost much more during the later stages of implementation and operation than during the earlier design stages. This is not only true of system composed through physical artifacts but also of software and knowledge intensive systems. Thus there is paramount emphasis on evaluating design alternatives to a high degree of confidence before moving on to the implementation and operation stages. The design process can be expanded into a cycle of generating design alternatives, modeling and evaluating them against system design objectives until a satisficing alternative is chosen to be moved forward to the implementation stage (Figure 7). The approach to modeling and analysis is crucial to appropriately informing design and implementation. It is through the recommendations of the models, and the underlying mechanisms for translating a model's recommendations to real world implementation that effective designs are achieved. If this translation cannot completely be implemented, the modeling activity is not very useful. Moreover, the efficiency in translating conceptual alternatives to models and back determines the design cycle time, which ultimately affects the system designer's ability to analyze the transformed system in a timely manner in response to fast changing system demands.

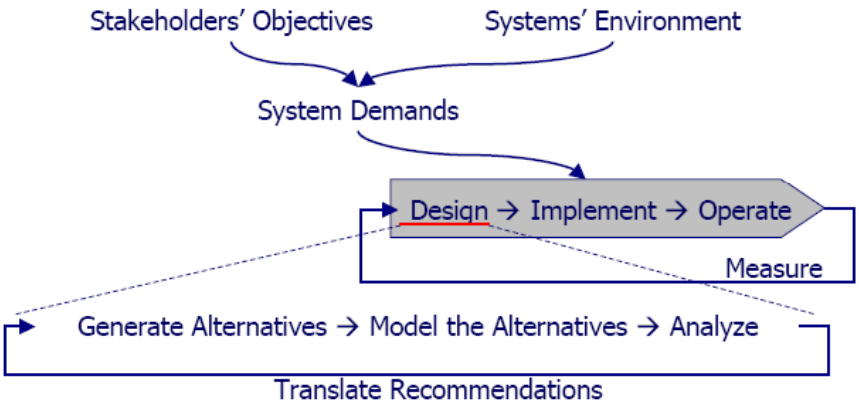

Figure 7: The design and re-engineering cycle

Most modern systems are being designed and transformed through the use of off-the-shelf and dedicated components that are designed by different vendors and over different periods of time. As the number of system components and their interactions increases, it becomes increasingly important that the models that evaluate the designs preserve the structure of these components and their interactions to prevent further complications and approximations in translating the models' recommendations to real world design changes. To ease complexity and increase efficiency, it is important that a designer be able to relate models directly with design components, and be able to replace them in a structure-preserving manner corresponding to the real world design activity.

Furthermore, the pace of technological change is much faster than the life of most systems and the gains in effectiveness and efficiency with the adoption of change are sometimes high enough to justify trading off existing components with the new ones. In such fast paced changes, having to completely revamp existing models of a system to accommodate a transformation made through few system components is not practical.

The system models have to be extensible, flexible and scalable. This further strengthens the requirement of a modeling approach that is structure preserving and can accommodate transformations in the same fashion as the real world, i.e., through replacement of existing components or addition of new ones.

Such requirements from modeling components have been very prominent in the software engineering community and have been addressed through a number of architectural paradigms that go beyond standardization of interactions, thus facilitating creativity while still providing for extensibility and change. One of these paradigms is the Model Driven Architecture (MDA) from the Object Management Group (www.omg.org).

Figure 8 shows a changed schematic of the system design and reengineering cycle from Figure 7. The primary changes to be noted are that modeling of design alternatives is now shown as a design composition activity with the use of models of design components that preserve their behavioral and interactive structure. Second, the recommendations of design are translated in terms of the components that make up the system and the system configuration, as compared to more abstract models such as probabilistic models. Beyond preserving the structure of the designed system with respect to design components, this approach also maintains a repository of design components that can be used off-the-shelf to compose newer systems. Any new components are also added to this repository for future use thus facilitating extensibility, reusability, flexibility and design scalability. A third point to note is that now there is a much more direct translation from design to implementation; in case of software systems, for example, such models directly generate most of the implementation code for the design alternative. By explicitly maintaining such models, the designer has added flexibility to change

implementation while keeping the conceptual models intact; in other words replacing the instance of a design component by a different, perhaps, better performing component that still conforms to the structural and interaction specifications of the modeled alternative.

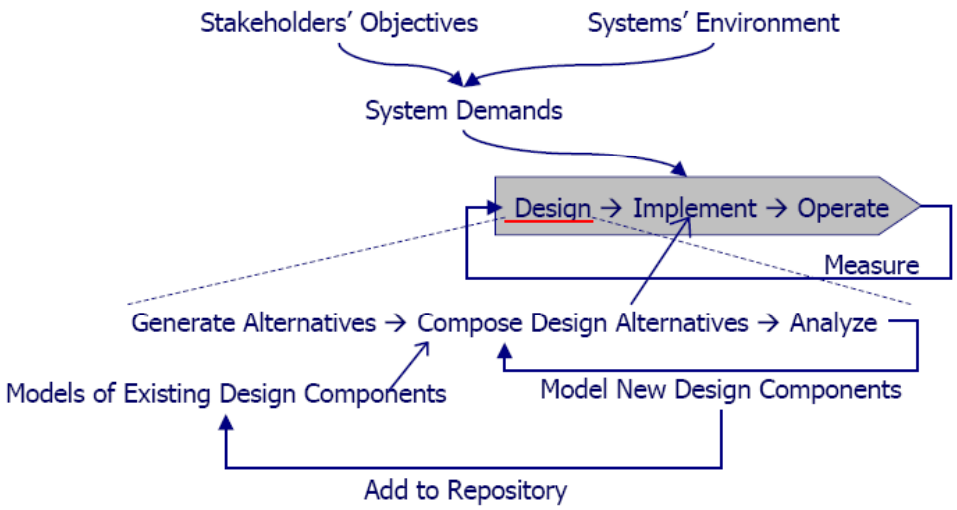

Figure 8: Model Driven Architecture

Recently the concept of Design Driven Architectures (DDA) has been introduced in the web-services and business process modeling world. The term was coined for promotion of the idea of deriving enterprise applications directly from business-process designs, i.e., design to deployment in one step (Ghalimi, 2002). Conceptually, design driven architectures are meant to enable a designer or analyst to alter the design of the system

using the design variables they use in the real world, for example, variables such as technologies and work-processes in socio-technical systems as compared to abstract translations such as percent change in effectiveness or efficiency of a sub-system.

When using a design-driven architecture the fundamental units of modeling correspond to the fundamental units of design in the real world. The intrinsic structure and interaction of design variables is preserved in the design evaluation models, thus design activity does not require additional translation from design concepts to implementation (Figure 9). Moreover, the nature and structure of system evaluation metrics corresponds one to one with the real world (pink area in Figure 9). That is, the terms in which the metrics are collected and evaluated in the model corresponds closely with the target real world system. Thus the evaluation and recommendations of the models semantically correspond to the real-world and do not require any abstract translations. These features help bridge semantic gaps between models and the real world design variables, thus naturally making the design tools easy to use for the designer.

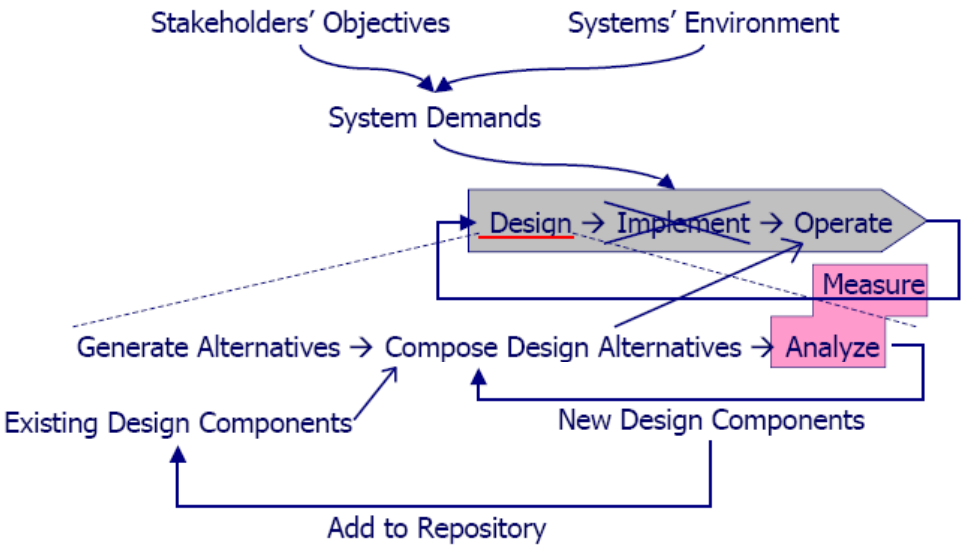

Figure 9: Design Driven Architecture

Not many instantiations of design driven architectures exist even in the software world. The Predictable Assembly from Certifiable Components (PACC) initiative at the Software Engineering Institute (CMU-SEI, 2004), is in my opinion, closest to achieving such efficiency in designing and implementing systems. To date PACC has only been employed in two real-world applications that relate to power-distribution sub-station automation and industrial robots (Hissam, Hudak et al., 2003; Hissam and Klein, 2004). These implementations primarily consist of a two-part design component, one corresponding to the implementation for the target system and one for simulation based analysis. Although a few other similar implementations exist (not referenced here because they are proprietary), such an approach to modeling systems for the purpose of design an implementation is fairly novel and rare.

This discussion was meant to give some insights into modeling approaches mean to facilitate efficient design and analysis. The software architecture and simulation platform

developed in this thesis builds up on these paradigms to ensure efficiency, scalability and extensibility of large-scale system designs, primarily because these systems evolve through transformations in specific parts of much larger systems.

2.4 Core Insights and Challenges

This chapter discussed relevant concepts from the field of cognitive engineering, agent-based modeling and simulation and modeling approaches for supporting design to aid the understanding of and to provide the groundwork for the theoretical and practical developments in this thesis. A number of useful insights were gained as summarized below:

1. System performance emerges from collective work of the workers, and this worker behavior arises from interaction with the work environment. Thus, to appropriately model work, both the work environment and the worker-activity must be represented, and the model should be sensitive to both these aspects.
2. Work environment centered transformations to socio-technical systems are implemented by changing the technology, processes and information in the system. These changes could be formulated in terms of changing the internal mechanisms of the instances of these components or changing the configuration of the system in terms of which components constitute it and how they are structured by work-relevant relationships. When analyzing such transformations, a work environment centered approach must be taken that emphasizes explicitly modeling the work environment such that models of the system can be changed in terms of these transformation variables.

3. Work environment constitutes all things, physical or not, in the system that affect the workers' choice of activities and the outcome of those activities, independent of any particular worker. Thus, physical objects, technologies, processes, information, and organizational structures are all a part of the work environment and must be considered when analyzing work.
4. Environment-centered models of work are based on the premise that workers employ knowledge of work-affording and work-constraining relationships amongst system components to accomplish their work objectives. This knowledge, when made available to them externally, supports them in their work activities but requires the worker to have a general inference and problem-solving model that can employ those external representations to identify the specific tasks which will achieve their goals.
5. Specific relationships between system components support specific kinds of work. Several such relationships exist in any real-world socio-technical systems and warrant modeling as appropriate to the work of interest to analysis. Means-ends relationships between environmental components and goals, topological relationships between physical objects, organizational relationships between roles of workers, and contextual relationships between the decision variables are all examples of such relationships. Existing environment-centered frameworks cater to one or a combination of, at most, two of such relationships, and are limited in the types of other relationships they can incorporate. Furthermore, there is a lack of approaches that integrate these frameworks and leverage the collective set of relationships that they model.

6. All existing work environment analysis frameworks employ descriptive models, thus limiting their applicability to analysis of small-scale systems and to subjective judgments. In contrast, computational models afford analysis of large-scale systems and quantitative comparison of design and transformation alternatives. Furthermore, computational models can be used in simulations to trace worker activity, giving detailed insights into the impact of transformations on both worker and system performance. Additionally, semantics of work-relevant relationships between system components can be represented declaratively, thus enabling semantic analysis of dependencies in the networks that are formed by these relationships.
7. A number of formulations from cognitive science, decision theory, dynamic systems and artificial intelligence can be used to model workers as agents. Such computational models can be employed in simulations to analyze and predict the emergent performance of the system.
8. Structure-preserving models foster scalability, extensibility, reusability, flexibility, ease-of-use and efficiency when designing and engineering systems.

These insights indicate the theoretical and practical basis for a conceptual framework for analyzing the impact of transformations. In essence, a conceptual framework needs to provide for models of both the work environment and the worker. The models should be able to address a heterogeneous set of work-relevant relationships among work environment components. These models should also be computational to enable simulations and network analysis. The models should be structure preserving to foster ease and efficiency of use. Beyond the conceptual framework, a software architecture

and a simulation platform to instantiate these models must also be created. To achieve these objectives, this thesis recognizes the following conceptual and practical challenges:

1. How can the existing cognitive engineering models and agent models be extended to analyze worker, component and network level transformations to a socio-technical system?
2. Which fundamental work-relevant relationships between the system components should be modeled to analyze the impact of such transformations?
3. What should be the basic form of a framework that enables modeling of heterogeneous work-relevant relationships to support a variety of work problems? How can such relationships be modeled coherently, i.e., integrated within one environment model, to analyze (and support) different kinds of work?
4. What kind of computational modeling constructs best enable computer simulations of the system using these work environment models?
5. What should be the nature of agent models interacting with these work environment models? What fundamental constructs can be developed and what assumptions can be made for modeling agents without limiting the possibilities in modeling workers and their internal dynamics?
6. How can one integrate the principles of Cognitive Engineering, Agent-based Modeling and Simulation and modeling approaches for system design to create a conceptual framework and simulation platform that, first, enable effective analysis, and, second, are easy and efficient to use?

The following chapters discuss this thesis' solutions to these challenges.

CHAPTER 3

THE CONCEPTUAL FRAMEWORK

This chapter develops the conceptual framework for modeling and analyzing worker, component and network level transformations in socio-technical systems. It develops modeling constructs for both the work environment and for the workers. This chapter builds on the conceptual underpinnings of cognitive engineering for models of the work environment and of agent-based modeling for models of the workers, and combines them to comprehensively model the system. The modeling constructs developed in this chapter allow for a system-level view, yet preserve the details of its structure at the level of its components and their internal mechanisms. Such an approach enables the designers to transform the system model through worker, component and network level transformations while analyzing its performance at the system level.

To model the work environment, this chapter extends the field of cognitive engineering which models the work environment through work-relevant descriptions. Since these descriptions maintain the system components' relevance to work, these representations are well suited for analyzing transformations in systems where workers' work and the system performance emerging from their collective work is the dynamics of interest, i.e., those systems of interest to this thesis.

Specifically, this chapter progressively extends current cognitive engineering methods through three extensions:

1. Currently, in cognitive engineering, the work-relevant descriptions of the work environment are combined with a representation of the problem-solving approach of the workers to analyze workers' interaction with their work environment. This thesis expands on this approach by, first, postulating that component and network level changes in a socio-technical system can be represented by incorporating those changes within cognitive engineering descriptions of the work environment: Component level changes can be reflected in models of component's work-relevant behavior and characteristics, and network level changes can be represented by changes in their work-relevant relationships with each other. Second, workers are assumed to use the same problem-solving approach within the current and transformed system. Thus, an analysis of the interaction between the workers and the representation of the transformed work environment should indicate the operational impact of the transformation, and an analysis of change of work-relevant interrelationships for network dependencies should indicate the network level impact.
2. By representing the structure of the system at the component, worker and network levels, this thesis tries to reduce the complexity of modeling and analyzing transformations made at these levels by preventing system designers from having to translate to different, more abstract formulations.
3. The descriptions are represented computationally to enable computational analysis of the emergent performance in the transformed system, thus enabling large-scale what-if analysis, analysis of large-scale socio-technical systems and transformations in them. More specifically, through computer simulations

designers and analysts will be able to visualize, predict and computationally analyze system performance that emerges from worker activities in the transformed system, i.e., perform an operational analysis. Through declarative modeling of components and their interrelations, system analysts will be able to identify network dependencies between them, i.e., perform network analysis.

This chapter starts with the development of these models for the work environment and then follows into a discussion about corresponding models of the workers. The chapter concludes with a summary of these conceptual developments and a discussion of how they meet the conceptual challenges listed in §2.4.

3.1 Modeling the Work Environment

Work environment consists of everything in the socio-technical system that affects the choice and outcome of work activities, independent of any particular worker. This definition includes all things, physical or not, that affect either or both the physical and cognitive activities of the worker. Therefore, physical objects, physical processes, technologies, procedures and regulations, information, and organizational structures are all components of the work environment. For example, the air traffic control system in Figure 10 consists of technological components such as the radar equipment, voice radio equipment and the aircraft, and process components such as the air traffic control procedures. The air traffic controllers (workers) interact with these components to accomplish their goals. Thus, the collection of all these components constitutes their work environment.

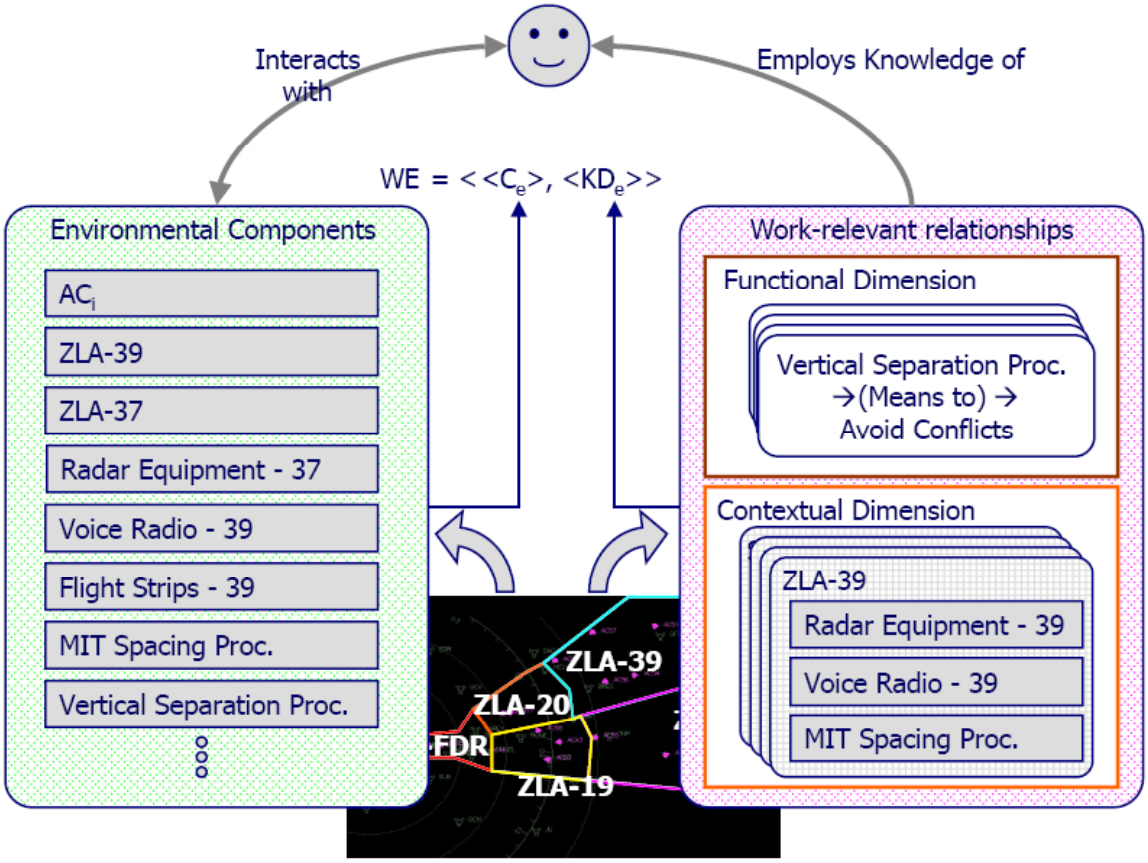

Figure 10: Elements of the work environment

Besides the collection of components, the work environment also includes the work-relevant relationships between these components. The workers employ the knowledge of these relationships to understand their work environment and to meaningfully interact with it. Examples of such relationships include the means-ends relationships that specify which components of the work environment can be used for the accomplishment of specific goals of the workers. For instance, Figure 10 illustrates that the vertical separation procedure (a work-process component) is a means to the achieving the goal of avoiding spatial conflicts between a pair of aircraft. Similarly, a different kind of

relationship can define how the workspace of a worker is configured by the components. For example, Figure 10 illustrates how components such as a radar, voice radio and procedures may comprise the workspace of the air traffic controller for sector ZLA-39.

Such relationships structure the work environment in work-relevant ways. A network of specific kinds of relationships defines one possible kind of structuring of the work environment, which can be viewed as one work-relevant dimension on which the components of the work-environment can be arranged. The work environment can be considered as a collection of many such dimensions.

Based on the above discussion, the work environment can be considered as a collection of the components and the dimensions of work-relevant relationships between them. These are the core constructs of the work environment. The following subsection develops the models of these core constructs and then builds up the model of the work environment using those constructs.

3.1.1 Modeling the Core Constructs of the Work Environment

This subsection starts with developing a model of an environmental component and then follows into modeling both the work-relevant relationships between them and the dimensions that they form. As discussed in the preceding section, an environmental component is anything, physical or not, that affects the choice and/or outcome of work-activities, independent of any particular worker. This definition includes all physical, technological, process and information elements of the work environment that are relevant to the work being analyzed (for example, Figure 10). Each of these work-relevant environmental components affects work by either constraining it or aiding it in

some way. Some components can be used in more than one way and towards more than one goal. For example, a rope can be used to tie up things if its length and strength permit, and it can be used to measure things if its length is known and does not change significantly during the act of measuring. In this example, *tying up something* was one goal, while *measuring* was another goal. Both these goals could be achieved with the use of the same component in the work environment, albeit in different ways. Whereas the rope's *length* was a common property useful for both the activities, *strength* is a property that does not significantly affect the latter activity. Similarly, the rope will also have some color, but this property is not important for any of the activities or goals discussed here so it would not be considered as a work-relevant attribute. It should also be noted that both activities are constrained by the length of the rope. If the length of the rope is not greater than the periphery of things being tied up, it cannot be used. Similarly, if the length of the rope is smaller than the things being measured, some other kind of aid or a more elaborate process is required to accurately measure using the rope. This example makes some interesting observations about the components of the work environment and how they relate to work:

1. Each component should be useful for at least one goal.
2. Each component has several properties, subsets of which may be needed for some activities while not for others.
3. Each component may be employed in more than one way, subsets of which relate to some kinds of work while not others.

Based on these observations, any component of the work environment can be viewed as a container of properties and of ways in which to employ it (usage mechanisms). These

attributes provide the components interface with the workers and the other components and can therefore be considered as its interface elements. A component may also have internal dynamics that govern how its properties change in response to the ways in which the component is used and immediate environmental conditions. These internal dynamics affect the outcome of the usage of the component, and therefore affect work. These internal dynamics therefore also require inclusion in the work-relevant model of the component. These internal dynamics govern the behavior of the component and are therefore referenced to as the behavioral elements of the component model. Thus an environmental component can be modeled as:

$$C_e = <ID_c, <P_c>, <UM_c>> \qquad \text{Model (1)}$$

Where,

C_e is the environmental component,

ID_c are the work-relevant internal dynamics (behavioral elements) of the component,

P_c is a work-relevant property of the component,

$< P_c>$ is the set of all work-relevant properties of the component,

UM_c is a work-relevant usage mechanism,

$<UM_c>$ is the set of all work-relevant usage mechanisms of the component, and

<> operator specifies a set of the enclosed symbols.

Figure 11 illustrates this model through the example of a physical/technological component, the 'radar equipment – 39', and of a work-process component, the lateral separation procedure. As shown, the internal mechanism by which the radar equipment sweeps the airspace every twelve seconds constitute its internal dynamics. This is the

behavioral element of the radar's model. Additionally, the radar equipment provides the radar blips of all aircraft in this airspace as a property that can be read by the air traffic controller. The radar also has certain usage mechanisms that help the air traffic controller in employing the radar such as a mechanism to read the coordinates of the aircraft. It is through using these interface elements (properties and usage mechanisms) that the air traffic controller interfaces with the radar equipment. To successfully interact with the radar equipment, it is both necessary and sufficient to know about the interface elements of the component in the context of their relationships to the work of the worker (discussed further in the following paragraphs).

Figure 11 also shows a work-process component, a lateral separation procedure, which is static in nature, i.e., it does not have any internal dynamics that changes the values of its properties. However, the work-process does have properties that expose the enclosed process and an expression of the applicability of that process. Similar to the radar equipment, it includes a set of usage mechanisms that specify how this component may be used.

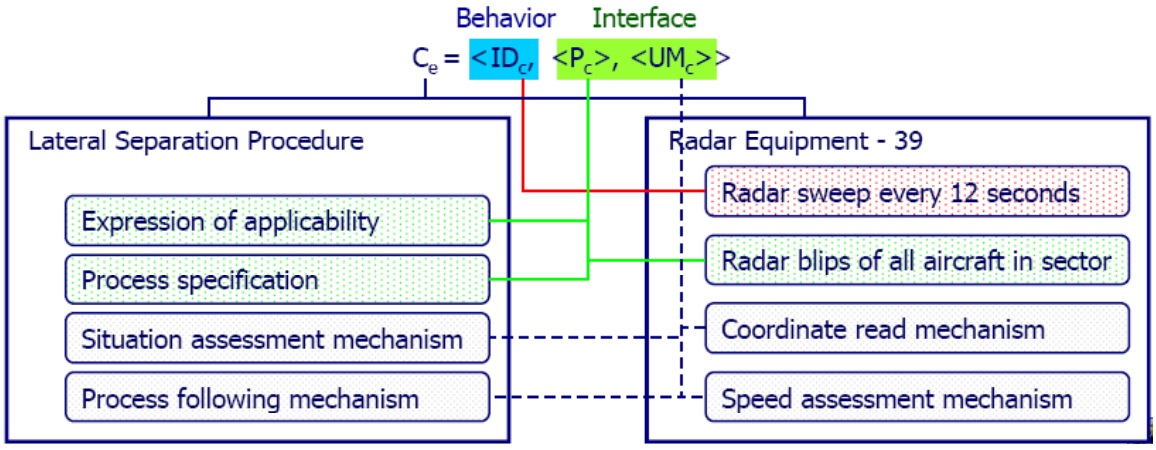

Figure 11: Example of model elements of environmental components

The work environment can be modeled (at a minimum) as a collection of such work-relevant component models, with all their work-relevant internal dynamics, properties and usage mechanisms included. However, as discussed in the preceding section, such a model of the work environment would not be complete for the purpose of supporting or analyzing work: it would not establish any environment level relationships with work, but would instead be limited only to relationships at the level of individual components. For example, how would a worker know which of the components in the collection of components are within his or her workspace, or which of the components should or should not be used towards specific goals? Such work-relevant relationships between the components in the work environment also need to be included in the model of the work environment, both for analysis of this model to be useful for the workers to interact with the work environment as a whole, and network level transformations that cannot be understood by only modeling the work-relevant relationships at the component level.

Components in the work environment can have a number of work-relevant relationships between them that structure the work environment in meaningful ways. For example, a specific kind of relationship defines the workspace of a worker. An example of such a relationship, shown in Figure 10, specifies that the workspace of the air traffic controller of sector ZLA-39 includes the radio equipment, the voice radio and the specific procedures that are available to that controller. Many instances of such relationships can exist in the same system, each defining one of many possible configurations for workspaces of different air traffic controllers. The set of these instances of this 'workspace' relationship forms a 'network' that defines one dimension of the work environment's structure that is useful in understanding the work environment and for the accomplishment of work. For example, air traffic controllers use this knowledge of their workspace to explore the space of possible components that they can interact with to accomplish their work. Similarly, different kinds of relationships may associate components of the work environment to the goals of the workers, or may associate the physical components with each other through spatial relationships.

The networks formed by each of these kinds of relationships structure the same work environment in one work-relevant way, and the knowledge of each dimension is used by the worker to do some specific kind of work. This thesis calls such each network of relationships a 'knowledge dimension' (or simply a dimension) of the work environment, and models the work environment as having a collection of one or more such knowledge dimensions.

While each dimension enables the worker to do some specific kinds of work, a combination of these dimensions can further be used to do work that cannot be done with

the knowledge of any one dimension alone. For example, in air traffic control, finding a useful and usable work-process to avoid conflicts requires the knowledge of the dimension that specifies that a lateral separation procedure is a means to this goal (functional dimension in Figure 10), and the knowledge of the dimension that specifies whether this procedure is available in the workspace of the worker (contextual dimension in Figure 10). The model of the work environment includes these work-relevant dimensions, along with the collection of components, to model the multiple networks in the work environment. The model of the work environment is therefore represented as:

$$WE = \langle\langle C_e \rangle, \langle KD_e \rangle\rangle \qquad \text{Model (2)}$$

where,

WE is the work environment,

C_e is a component,

$\langle C_e \rangle$ is the set of all work-relevant components in WE,

KD_e is a knowledge dimension, and

$\langle KD_e \rangle$ is the set of all work-relevant knowledge dimensions in WE.

It may be argued that, since the worker employs the knowledge in the dimensions to do his or her work, these dimensions should be a part of the worker model. Placing this knowledge in the work environment is justified, however, through noting the nature of transformations in a socio-technical system. For example, what would happen if the configuration of components in the workspace of the worker were changed? If the knowledge dimensions were embedded in the model of every worker, they would not be able to find a useful work-process in this transformed work environment unless the models of each were updated with every change to the work environment, even though

the transformation was enacted in the work environment. In contrast, if this knowledge is represented externally and the worker employs a general knowledge-processing capability that can scan through the dimensions of the work environment, the worker will be able find and complete the work-process in the changed work environment. At a conceptual level, the correct abstraction will be changed in response to changes in the work environment, and, at a practical level, the change would need to be made in only one place.

With this model form, the work environment can be viewed as a multidimensional space where each component is represented on one or more dimensions. Recalling insights 3 and 5 in §2.4, each of these dimensions supports specific kinds of work. With the addition of more dimensions, the model supports more types of work. Unlike current cognitive engineering models, there isn't any conceptual limitation on the number of dimensions that can be added into this model of the work environment, and there is no limitation on the nature of relationships in any dimension; thus, this model form is extensible for supporting different kinds of work. This model form thus allows for more comprehensive analysis of complex socio-technical systems.

This model form thus tackles the first issue of challenge 3 in §2.4, i.e., development of a basic model form capable of incorporating heterogeneous work-relevant relationships to support different kinds of work.

To tackle the second issue in challenge 3 in §2.4, i.e., modeling relationships in a consistent way across the dimensions, let us look at the basic constructs that are common to all dimensions. Each dimension has a number of instances of work relevant relationships, where each instance associates a set of components. Each relationship is

also described by certain parameters, the values of which fully define the instance of the relationship. For example (Figure 12), the functional dimension associates an environmental component with a goal of the worker through a means-constraints-ends relationship. To fully define this relationship it lists two parameters: MCF (Means-Constraints Flag) that identifies the component as a means (instead of a constraint) to the goal 'Avoid Conflict'; and the CAT parameter (Category Specification) that identifies the category of the component as being a 'work-process'. Thus a knowledge dimension is modeled as:

$$R_d = <[C_e], <P_r>>, \text{ and} \qquad \text{Model (3)}$$

$$KD_e = <R_d> \qquad \text{Model (4)}$$

where,

Rd is a relationship,

C_e is one of the environmental components that is associated with others by the relationship,

$[C_e]$ is the subset of the environmental components that are related by the relationship,

$<P_r>$ is the set of parameters needed to fully define an instance of the relationship, and

KD_e is a knowledge dimension, i.e., the set of instances of all relationships of a particular type.

Figure 12 illustrates a multidimensional model of the work environment, in this case containing the functional and contextual dimensions. Each dimension encloses numerous work-relevant relationships of the same type. For example, the functional dimension is

the set of all means-constraints-ends relationships between the components of the work environment. Each of these relationships associates a goal of the worker with the component acting as a means or a constraint to that goal. For example, the vertical separation procedure is a means to the goal of avoiding a conflict. The contextual dimension, on the other hand, associates a set of components such that they define the configuration of a worker's workspace, i.e., it uses a contextual composition relationship between the components of the same work environment. For example, this dimension identifies the workspace of the air traffic controller of sector ZLA-39 as consisting of the radar screen, the voice radio and a set of control procedures (Figure 12).

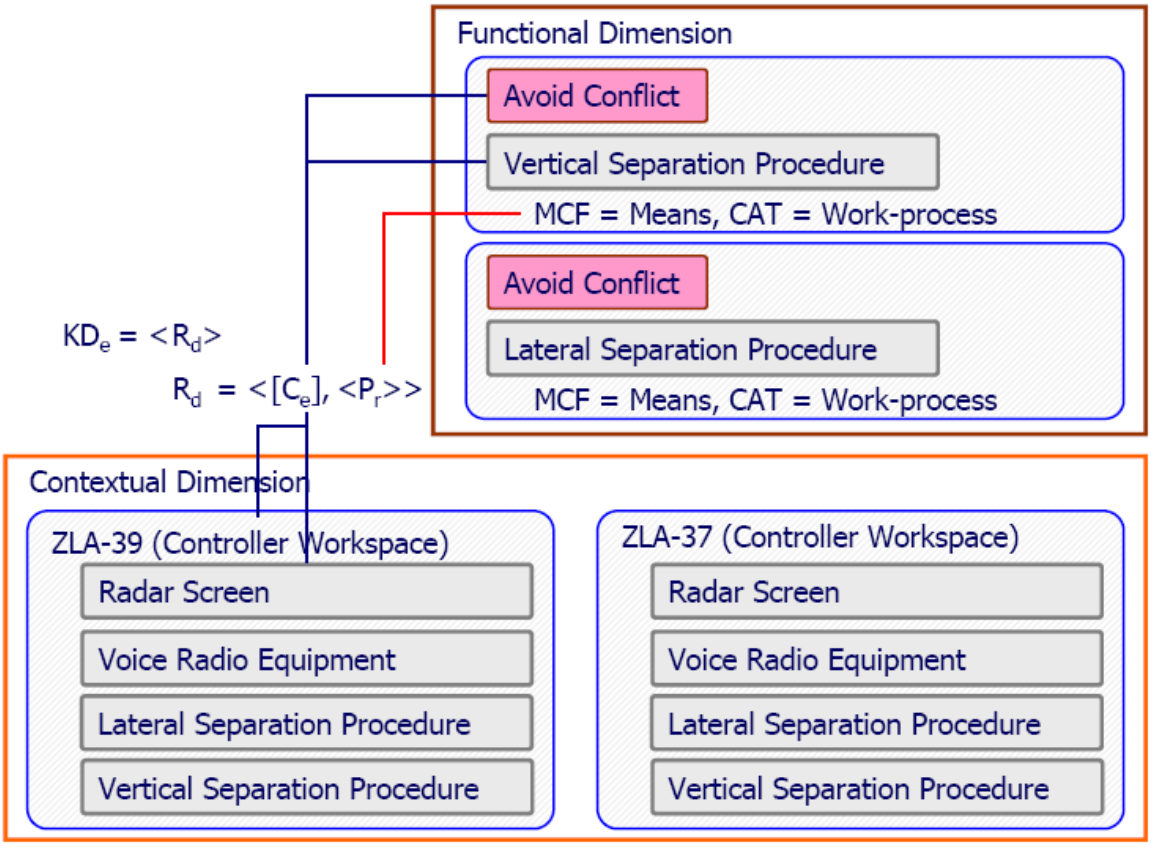

Figure 12: Example of knowledge dimensions

Now that we have established the models of the work environment, its dimensions, and its components, we need to look at how the model of the components relate to a knowledge dimension. In other words, how does a component map onto a knowledge dimension? Figure 13 illustrates this mapping by taking an example of the contextual dimension that is used to specify the configuration of the workspace of a worker. Shown here are the contributions of two environmental components to the workspace of the air traffic controller of sector ZLA-39. This workspace consists of a radar screen, voice radio equipment and two separation procedures (work-processes). When interacting with

the radar equipment the air traffic controller only needs to interact with the radar screen's radar blips of the aircraft (properties) and its usage mechanisms for reading the coordinates of the radar blips and estimating aircraft speeds. Thus, only this subset of the interface elements of the radar equipment are needed to associate it with the contextual dimension. This subset is the dimension-relevant interface of the component and represents the dimension-specific view of the component. In this thesis, this view is termed as an 'aspect' of the component.

Taking a parsimonious approach to prevent modeling of unnecessary interface elements in the component and to enable a multidimensional model of the work environment, only those interface elements of an environmental component should be modeled that correspond to the set of dimensions used in the work environment model. Thus a component can be modeled as:

$$C_e = \langle ID_c, \langle CA_c \rangle \rangle \qquad \text{Model (5)}$$

$$CA_c = \langle [P_c], [UM_c] \rangle \qquad \text{Model (5a)}$$

Where,

C_e is the dimensional view of the component,

ID_c are the internal dynamics not visible to any knowledge dimension,

CA_c is an aspect within the component relevant to a knowledge dimension,

$[P_c]$ is the subset of component properties (refer Model (1)) relevant to the aspect's knowledge dimension, and

$[UM_c]$ is the subset of component usage mechanisms (refer Model (1)) relevant to the aspect's knowledge dimension.

It should be noted that a component's aspects that map onto different dimensions could possibly have an overlapping set of properties and usage mechanisms.

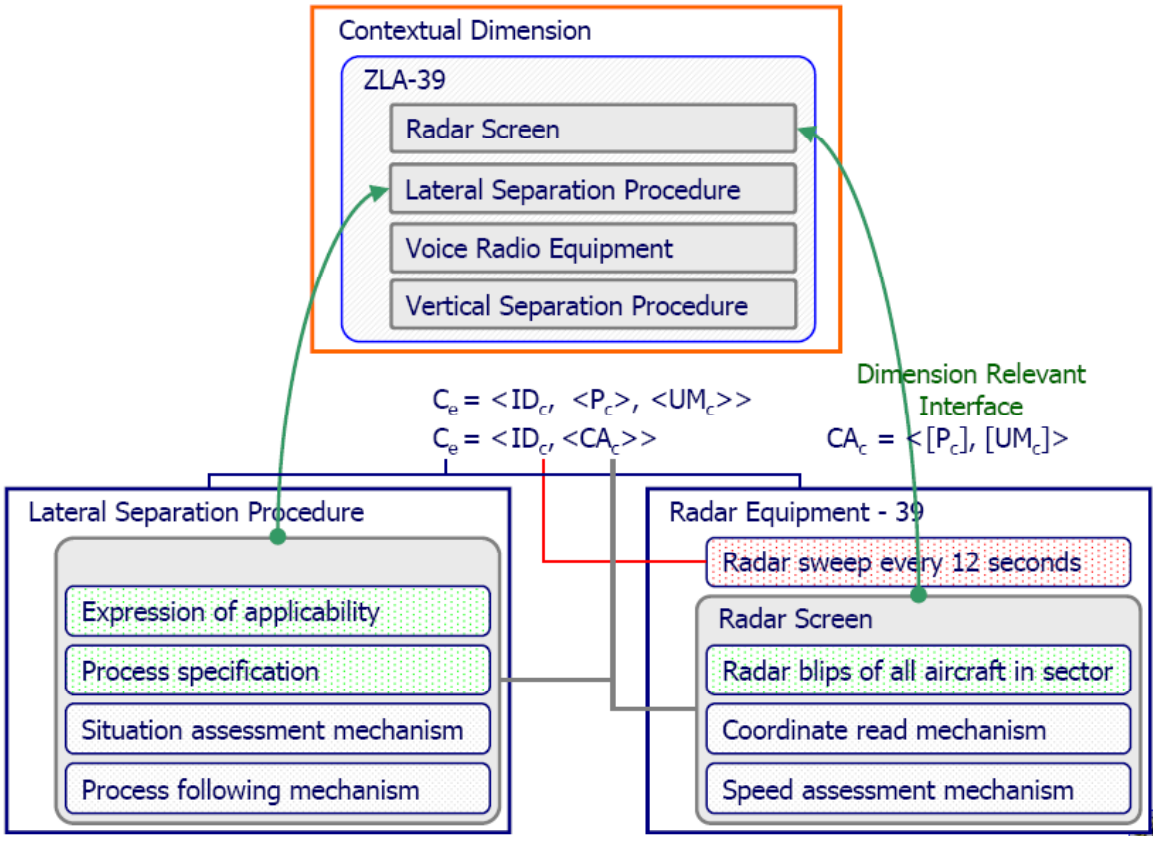

Figure 13: An aspect is a dimension relevant view of the environmental component

Conceptually, identifying aspects maintains a clear view of the components in the work environment in terms of how they are used by the workers and what parts of them are necessary and sufficient for the dimensions in the model. They provide for a natural

grouping of attributes that relates to the nature of work being analyzed, and the nature of relationships between transformation variables at the component and the network levels. They thus provide for structure-preserving modeling of the work environment with natural associations between different levels of transformations. §3.2 will show how this construct also relates the component and network level transformations to the worker level transformations and the construction of the worker models.

Aspects also provide several utilitarian functions:

1. The same component can be linked to or removed from different dimensions by adding or removing aspects relevant to those dimensions and linking them with the aspects of other components through dimension-specific relationships. Thus this mechanism provides the foundation for building the multidimensional work environment model, where each dimension is a network formed by the relationships therein.
2. By explicitly grouping model elements of a components in relevance to a dimension that establishes environment level work-relevant models of the work environment, an aspect makes it possible to structurally relate component-level and network-level transformations, and thus assess structural implications of changes at one level on the other, i.e., network analysis (described further in §4.6).

In summary, this representation of the work-environment establishes a basic, generalizable, scalable, extensible and work-relevant model. This formulation can be used with models of the workers that both use problem-solving algorithms referencing the work environment and account for internal capabilities and skills of the workers. The

collection of worker-models and the work environment model describes the complete system in a manner that can be used to analyze the impact of transformations at the component and the network levels. Table 2 summarizes these core constructs. The next subsection discusses the fundamental dimensions that should be modeled for work-relevant analysis of transformations.

Table 2: Summary of the work environment model

Model	Description
$WE = \langle\langle C_e \rangle, \langle KD_e \rangle\rangle$	Work environment: The work environment is modeled as a collection of all the work-relevant components in the system and all the dimensions of knowledge of relationships between these components that the workers need to accomplish their work.
$KD_e = \langle R_d \rangle$	Knowledge Dimension: A knowledge dimension is a set of work-relevant relationships that associate components in the work environment. A knowledge dimension includes all instances of these relationships and thus establishes one kind of structure or network in the work environment.
$R_d = \langle [C_e], \langle P_r \rangle\rangle$	Work-relevant Relationship: A relationship associates components in the work environment in work-relevant ways. The knowledge-processing engine in the worker model is capable of parsing relationships to derive the model of a task.
$C_e = \langle ID_c, \langle P_c \rangle, \langle UM_c \rangle\rangle$ $C_e = \langle ID_c, \langle CA_c \rangle\rangle$	Environmental Component: Physical objects, processes, technologies, procedures and regulations, information and organizational structures that affect worker activities. This model contains their internal dynamics, work-relevant properties and usage mechanisms, which may be grouped by knowledge dimension into a set of component aspects.
$CA_c = \langle [P_c], [UM_c] \rangle$	Component Aspect: The component aspect is the view of the component from one knowledge dimension.

3.1.2 The Fundamental Dimensions of the Work Environment

A number of dimensions may be needed to model the work environment. The constructs given in the preceding section were purposefully stated generally so that any number of dimensions may be included. This section details two 'fundamental' dimensions that are broadly applicable.

As defined in the previous chapter (§2.1.1) and this chapter: (1) work activities are directed towards achieving some goals, and (2) work-activities are 'situated', i.e., their execution is contingent to the current state of the work environment. Thus there are two primary relationships that should be modeled when modeling work environment of the socio-technical system: (1) the functional dimension that relates the components to the goals, and (2) the contextual dimension that defines the workspace of the worker in terms of which components are accessible to the worker by virtue of the worker's placement in the work environment. The following sub-sections detail these two dimensions.

3.1.2.1 The Functional Dimension

The functional dimension relates the components to the workers' goals. Building on the theoretical underpinnings of cognitive engineering and topological psychology, this dimension describes each component as an affordance to or a constraint on achieving worker goals (Gibson, 1979; Gibson and Crooks, 1938; Lewin, 1936; Marrow, 1969; Rasmussen, 1985; Rasmussen, 1988; Rasmussen, Pejtersen et al., 1994; Vicente, 1999).

The one relationship that constitutes this dimension associates two entities: a worker's goal and an environmental component of any type. This relationship is termed here as the 'means-constraints-ends' relationship. This is a directional relationship that

originates from the component and ends at the goal. The relationship has two parameters; one flags whether the component affords or constrains the goal, i.e., whether it is a means or a constraint for the end, and the second identifies the category of the component. A component's category is required to enable the worker in finding the right kind of component, for example, a 'procedure' or an 'information system' that enables the achievement of a goal. It is assumed that the worker is equipped with the knowledge or general mechanisms to employ components of specific categories, thus governing the choice of which category of component to use. This parameter is therefore used by workers to narrow their return set when searching for a useful component in the work environment. Thus, in terms of model (3) and model (4) in §3.1, the relationship may be represented as:

$$SC_e = \langle C_e, G \rangle, \qquad \text{Model (6)}$$

$$\langle P_r \rangle = \langle MCF, CAT \rangle, \qquad \text{Model (7)}$$

$$MCER_{fd} = \langle SC_e, P_r \rangle \qquad \text{Model (8)}$$

$$FD = \langle MCER_{fd} \rangle \qquad \text{Model (9)}$$

Where,

C_e is any environmental component as represented by Model (1),

G is the goal with which C_e is associated, and is itself also a component of the work environment,

SC_e is the pairing of any environmental component and a goal,

MCF is the means-constraints flag, a parameter of type P_r (refer Model (3)) used to define the $MCER_{fd}$ relationship,

CAT is the category of the component, another parameter of type P_r (refer Model (3)) used to define the $MCER_{fd}$ relationship,

$<P_r>$ is the set of relationship parameters used to define the $MCER_{fd}$ relationship,

$MCER_{fd}$ is the means-constraints-ends relationship of type R_d (Model (3)) of the functional dimension parameterized by $<P_r>$, and

FD is the functional dimension of type KD_e (refer Model (4)).

The purpose of this dimension is to help the worker perform the following queries when solving work problems:

1. Which components in the work environment afford or constrain the achievement of the given goal?
2. Which components of a given category (CAT) afford or constrain the achievement of the given goal?
3. Does a given component afford or constrain a given goal?

From the results of these queries the workers' problem-solving engine will generate the work task that employs components appropriately. For example, by executing the second query, the worker can obtain a procedure for achieving a particular goal. The problem-solving algorithm can then invoke a procedure processing capability in the worker capable of reading and executing its actions, thus deriving the work tasks.

3.1.2.2 The Contextual Dimension

The contextual dimension describes the workspace of the worker in terms of which components are accessible to the worker and which are not by virtue of the worker's placement in the work environment. As noted in §3.1.1, environmental components have a set of work-relevant properties and a set of work-relevant usage mechanisms. It is

through these properties and usage mechanisms that a worker directly interacts with the components. However, the worker can only consider properties and usage mechanisms that are accessible to him or her, and then must have knowledge of where to find them. The contextual dimension defines this access structure, which may change dynamically during the lifetime of the system, including changes when the workers change their placement in the work environment.

The contextual dimension contains two relationship types. The first relationship type groups subsets of the attributes of one or more components in a 'contextual node' which, when the node is accessible to a worker, provides access to those attributes of all components within it. The second relationship type, 'contextual-composition', groups a set of contextual nodes in a hierarchical fashion, making all contextual nodes downstream of the hierarchy accessible to the worker while keeping the upstream contextual nodes inaccessible. Figure 14 pictorially represents this relationship. Each contextual node in the picture is a collection of aspects of components that group a subset of their relevant properties and usage mechanisms. The arrows between these nodes represent the contextual-composition relationship. A contextual node at the head of the arrows is composed of all contextual nodes at the tails of the arrows. Thus, any worker having access to contextual node 1 in Figure 14 also has access to all downstream contextual nodes, i.e., nodes 2, 3, 4, and 5. However, a worker having access to node 2 but not node 1 has access to only nodes 2 and 5. The contextual-composition relationship can specify any number of such hierarchies with any depth and containing any number of contextual nodes. There can be more than one disconnected hierarchy in this dimension, any

number of workers can access a contextual node, and any number of contextual nodes can be made accessible to a single worker.

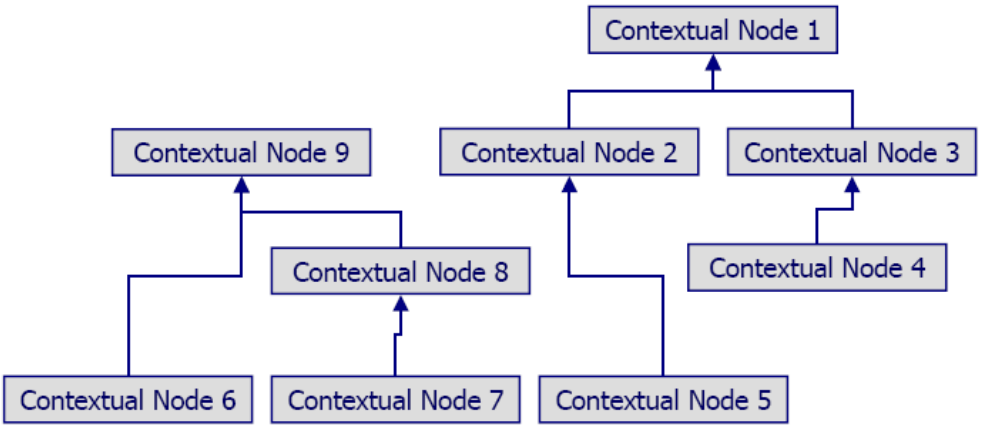

Figure 14: Pictorial representation of the contextual dimension

Using model forms (3), (4) and (5), this dimension can be represented as:

$$CN_{cd} = [CA_{cce}], \quad \text{Model (10)}$$

$$CC_{cd} = <CN_{cdp}, CN_{cdc}>, \text{ and} \quad \text{Model (11)}$$

$$CD_{KD} = <CC_{cd}, CN_{cd}> \quad \text{Model (12)}$$

Where,

CA_{cce} is an aspect of a component (refer Model (5a)) that contains the component attributes relevant to this knowledge dimension,

$[CA_{cce}]$ is a subset of all aspects on all components in the environment,

CN_{cd} is a contextual-node relationship of type R_d (refer Model (3)),

CN_{cdp} is the parent contextual node of type CN_{cd},

CN_{cdc} is the child contextual node of type CN_{cd},

CC_{cd} is a contextual-composition relationship of type R_d (refer Model (3)), and

CD_{KD} is the contextual dimension (refer Model (4)) containing all CN_{cd} and CC_{cd}.

The contextual dimension enables the worker to query the following:

1. Which components are accessible in a given contextual node?
2. In which contextual nodes is a component accessible?
3. Is a given interface element on a given component accessible to a worker who has access to given contextual nodes?
4. Is a given interface element accessible to a worker who has access to given contextual nodes?

Thus this dimension helps the worker to 'situate' him or herself in the work environment. With the combination of this knowledge and the functional dimension, a worker can find accessible components providing affordances for accomplishing a particular objective.

Appropriateness of this model form can be justified through proving that, first, the model form does not constrain the designer from designing specific kinds of contextual structures, and, second, by showing that the model form corresponds to the way the real world contextual structures are designed.

The first justification becomes apparent by simple application of the principles of set theory. Since a work environment designer is free to compose a set (contextual-node) in any manner and further compose supersets (contextual-composition) from those sets in any manner, this model form does not overly constrain the designer, and instead provides a systematic representation useful for a broad range of designs. Furthermore, different

supersets (hierarchies formed by contextual-composition) can have the same attribute represented in them.

The second justification stems from the process of designing and transforming work environments. As discussed in §2.3, socio-technical systems are composed from design components. Thus, the system is naturally built in the form of sub-systems and sub-sub-systems and so on. This compositional relationship for composing the contextual dimension is structure-preserving and captures a method of composition and decomposition intrinsic to the design of large-scale socio-technical systems.

3.1.3 Representing the Work-Process Component

This subsection illustrates the model of a work environment component with the example of a work-process component. As previously discussed, the work environment includes all things, physical or not, that affect the work of the workers. One such entity is a work-process, i.e., a structured ordering of worker-activities, the execution of which is meant to accomplish specific goal(s).

In most socio-technical systems, such as air traffic control and manufacturing plants, a number of work-processes exist. They are a part of the system irrespective of the individual(s) who employ(s) the work-process. Work-processes exist in the system to represent natural task-sequences discovered by workers or as regulatory procedures intended to normatively direct them. Just like any other work-relevant component of the work environment, a work-process helps in achievement of some goals but, in some conditions, it may constrain other goals, simply because the outcome of one or more of the activities in the work-process may put the system in a state that limits achievement of

other goal(s). It can be thus both an affordance for some goals and a constraint for the others. This knowledge is essential for a worker to correctly employ the work-process. Based on the discussions in §3.1.2.1, this component would therefore be modeled on the functional dimension. When modeling it on the functional dimension the CAT property in model (7) would be symbolized to indicate that it is a work-process, and the MCF flag would represent whether it is a means or a constraint for the associated goal.

Not all work-processes are available to all workers in a system. For example, in air traffic control work-processes to separate aircraft are available to aircraft controllers and not pilots. This distribution is controlled through the contextual structures that the workers are placed within. For example, the work-processes may be made available through the use of an information system or a task guidance system available only to a few people. In such a situation, the component can be appropriately modeled on the contextual dimension.

Examining the model of the component itself, model (1) identifies three parts to a component: internal dynamics, a collection of properties and a collection of usage mechanisms. A work-process does not have any internal dynamics and the first part is therefore empty. Let us now examine a work-process' properties and usage mechanisms. Usually, the work-processes are only applicable in certain situations. Each situation is specified with respect to a specific contextual structure; thus each work-process' properties may include a specification of the contextual nodes and their states within which it applies. A work-process' properties also include the specification of the process itself, i.e., the structured ordering of the activities.

Thus the work-process has two properties: one specifies the situation in which it is applicable and the second specifies the process. These two properties assume a usage mechanism that can read these properties and follow the process. For example, the usage mechanism should be able to parse the expression of the situation, assess the work environment for the values of the contextual variables and, to the extent the real worker is capable of assessing the appropriateness of the process in context, verify whether the work-process is appropriate for the contextual structures. Similarly, there should be a usage mechanism that executes the process and takes appropriate actions on other components in the work environment.

It should be noted that this component imposes certain requirements on the competencies of the worker using the component:

1. An ability to process the functional dimension;
2. An ability to process the contextual dimension; and
3. The capability to employ the usage mechanisms of the component.

Such competency requirements are used to construct models of agents, as discussed in §4.6.

3.1.4 Discussion

In this section a multidimensional model of the work environment was constructed. This model is capable of incorporating multiple heterogeneous work-relevant relationships between the components of the work environment (i.e., challenge 3 from §2.4). Through the establishment of dimensions, the framework is also suitable for representing a network of relationships between a set of components. Thus, this model is suitable for

analyzing network level transformations made over different kinds of networks in the same work environment. Additionally, Models (1) and (5) of the environmental component are well suited for analyzing component level transformations by allowing for changes to both the interface(properties and usage mechanisms) and the behavior (internal dynamics) of the component. This section also presented the functional dimension and the contextual dimension as fundamental dimensions applicable to analyzing work-relevant transformations in most socio-technical systems (i.e., challenge 2 of §2.4). Finally, this section exemplified the model of an environmental component through modeling a work-process component.

3.2 Modeling the Worker

This section models the workers in a manner that can be combined with the models of the work environment developed in the preceding section for analyzing transformations to the socio-technical system. A worker can be viewed as a collection of behaviors modeled a priori, where a behavior is the situated response of a worker in pursuance of his or her goals. The premise behind such a viewpoint is that, if all behaviors of all the workers in the system are known, then system performance will emerge from combining these worker models with the structure and dynamics of the work environment, thus requiring no further abstraction of the worker. However, due to the very large number of possible behaviors of each worker, compiling such a model would be impractical. Furthermore, creating such a map of behaviors for every possible situation may be impossible due to the inherent non-linear and complex nature of worker behavior that arises from their

internal processing and their capabilities and limitations. Therefore, in this thesis the worker is modeled as an agent as described in the next section.

3.2.1 Modeling the Worker as an Agent

From the insights drawn in §2.2, at a very abstract level an agent can be described as a structured collection of perceptual, cognitive and executive skills and capabilities. Besides these skills and capabilities, the worker also has internal processors. These internal processors run autonomously, and employ these skills, capabilities and the worker's subjective knowledge of the work environment in pursuance of worker's goals. Therefore, to appropriately model a worker and its behavior one must model both the internal structure of the worker, i.e., its perceptual, cognitive, and executive skills and capabilities, and the aspects of these internal components that relate to and interact with the work environment.

Models of internal capabilities and skills should be able to interact with the work environment models developed in the previous section. Thus, the worker needs problem-solving skills and capabilities to decipher work-relevant relationships and to derive the tasks needed to accomplish the goals. Since the relevant relationships are modeled in the dimensions of the work environment, the work environment imposes a requirement on worker skills and capabilities that must be modeled to appropriately represent a worker situated in it. These skills and capabilities could be intrinsic to the worker, or they could be cultivated through training the worker.

In essence, a worker can be viewed as a collection of skills, capabilities and processors (Figure 15):

Worker = <<Skills>, <Capabilities>, <Processors>> Model (12)

where,

<Skills> is the set of all skills of the worker that may have been cultivated through the lifetime of the worker,

<Capabilities> is the set of general capabilities that are intrinsic to a class of workers; for example, human workers are assumed to have certain kinds of long term and short term memory, and

<Processors> is the set of all autonomous and parallel processors that perform the internal processing of the worker model and exhibit their deliberative and reflexive behavior. For example, human workers are assumed to be equipped with cognitive, visual, auditory and motor processors.

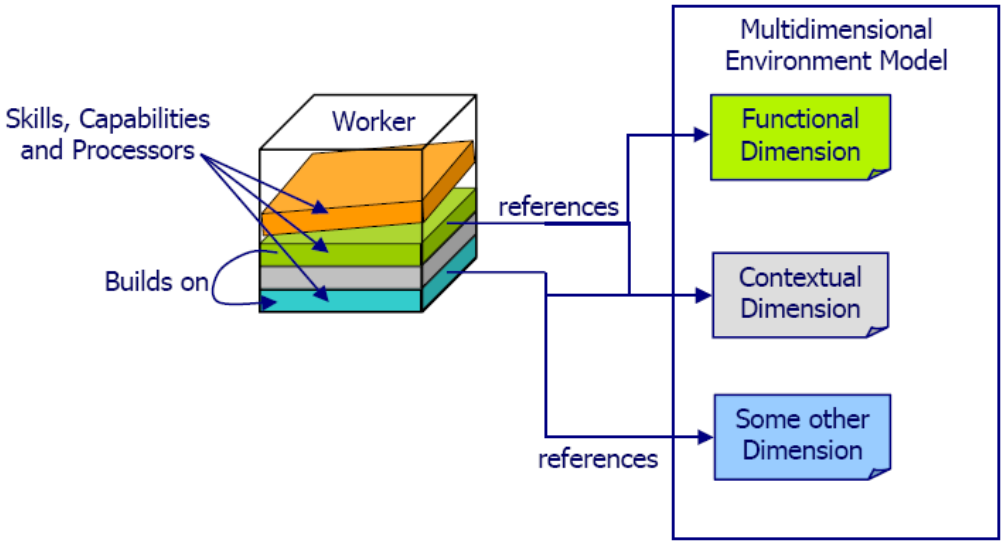

Figure 15: An illustration of the model of a worker

This framework requires the following of the agent models:

1. To address autonomy, the worker model must be equipped with at least one autonomous process that is guided by the goals and that manifests autonomy of worker behavior.
2. Worker capabilities and skills include those needed to process those dimensions of the work environment relevant to their work. The models of these capabilities and skills should be capable of employing the declarative knowledge in work environment models in semantically correct ways.
3. Each element of this collection of skills, capabilities and processors may algorithmically or structurally utilize other elements in the collection.
4. The internal architecture of the agent and can assume any form based on the broad underpinnings of agent-based modeling (§2.2). This thesis makes no assumptions about this structure to allow the designers to choose from the broad range of agent models that are available.
5. Analyzing work environment centered transformations requires associating worker skills and capabilities with the environment knowledge dimensions and components that the worker will employ. This association is required to perform a network analysis of the feasibility and requirements of the worker skills for a given configuration of the work environment. Thus the agent models must declaratively specify these associations when modeling a worker.
6. It should be possible to specify the worker model in terms of the skills, capabilities, and processors that constitute it and their dependencies on each other, irrespective of the underlying computational implementation of those skills,

capabilities and processors. This ensures that worker level transformations in behavior can be enacted by only referencing different computational implementations of isolated behaviors. Additionally, it allows for modeling individual differences in performance.

7. Worker models can be differentiated in terms of what capabilities constitute them, how they are structured, and which computational implementation they use.

The reader should note that these postulates do not impose any restrictions on the nature of the models of the agent's capabilities and their structural and algorithmic relationships to each other. It is these two facets of the model of a worker that are employed by different kinds of agent architectures such as artificial intelligence, cognitive and human performance models, thus leaving this aspect of the model open as appropriate to the domain and the needs of the analysis. A system analyst should be able to use any such models and architectures to analyze system performance with respect to different kinds of workers. For example, if an analyst wants to test system performance when a human worker is replaced with automation, the automation's model may have very different internal structure but should still be able to work with the same work environment. Such a model form separates the work environment from the internal mechanisms of the agents, thus ensuring portability of the work environment design to a variety of transformations and different kinds of workers. This distinction doesn't remove a system's sensitivity to the workers' limitations and activities, but instead establishes a clean boundary between what has to be known when modeling at the level of abstraction of the socio-technical system and what should be known to model internal mechanisms of

the agents. This also ensures that it is possible to separate transformations in the worker from transformations in the work environment.

3.2.2 Understanding Worker's Interaction with the Environment Model

To examine how a worker model will interact with the work environment, Figure 16 provides an example showing one dimension (contextual dimension) and one component (radar equipment) from the work environment model in the air traffic control example. To interact with this model of the work environment the worker first interacts with one or more dimensions of the work environment. From these dimensions the worker obtains the instances of the relationships that are relevant to his or her current work. The worker then parses these relationships to identify the components that it needs to interact with to do its work. The aspect that maps the component onto the dimension, for example, the radar screen that maps the radar equipment in sector ZLA-39 to the workspace of the sector controller, identifies the properties and the usage mechanism, i.e., the interface elements, of the component with which the worker interacts. Using his or her knowledge about parsing the relationships on a dimension and the general mechanisms to invoke the usage mechanisms on the components, the worker is able to successfully interact with the work environment in work-relevant ways. The invoking of the usage mechanisms on the components may trigger their internal dynamics, which can result in some localized or global response that may cascade through the system resulting in some system-level emergent dynamics.

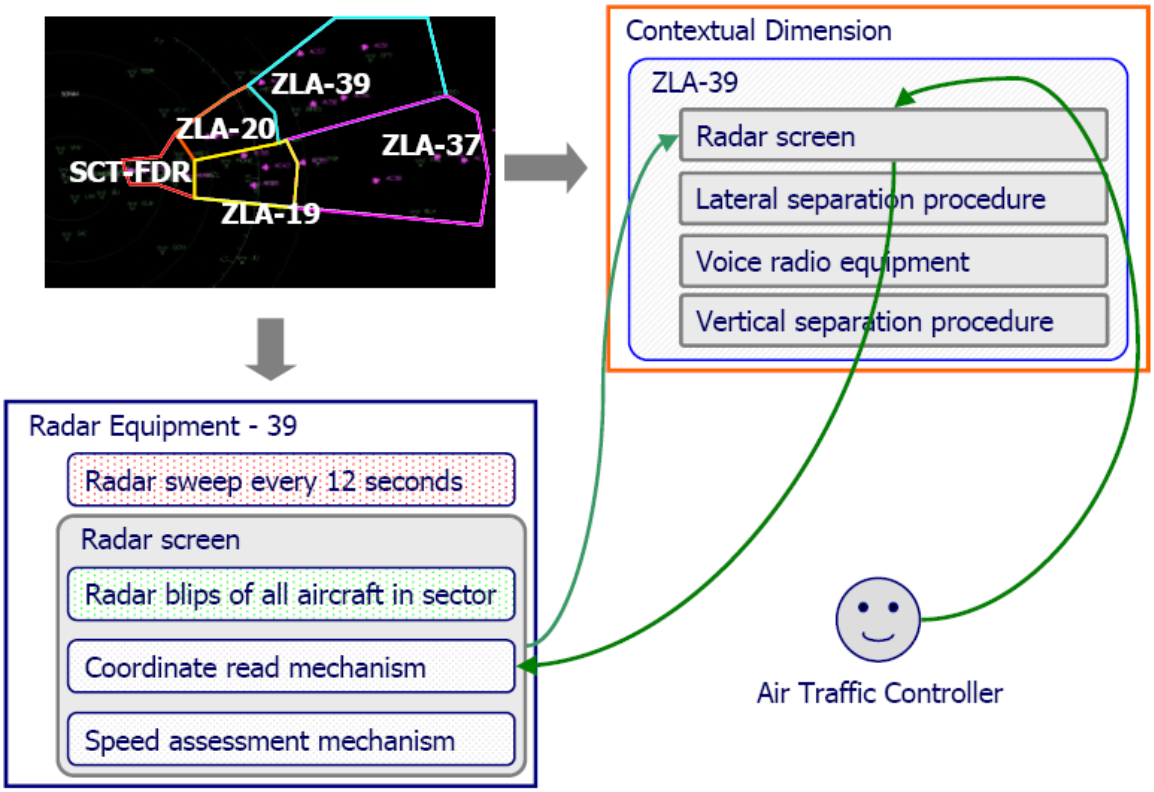

Figure 16: Agent's interaction with the work environment model

It should be noted that, for such interaction to occur, the workers should be equipped with some inference making and problem-solving approach that employs the knowledge in each or a combination of dimensions in the work environment model. This set of problem-solving approaches is specific to the modeled dimensions but can address any structure within those dimensions. Via the use of these problem-solving approaches, the worker parses the specification of the relationships and fits its components into the associated models of tasks that he or she should perform. This means that, given a multidimensional model of the environment, a worker will first sieve through it to

identify the dimension(s) that incorporate the work-relevant relationships of interest to his or her problem. Once each dimension is obtained, he or she will query it for a specific instance of a relationship based on certain parameters. The obtained instance of the relationship will then be parsed by the problem-solving algorithm within the worker to generate work tasks that the worker will execute to accomplish his or her work.

Therefore, using this model form, the problem solving algorithms within the workers must have certain common characteristics. First, they all must have task generation mechanisms that (1) are dedicated to the relationships in particular dimensions and their combinations and (2) can also address the constraints of skills and capabilities of the worker. Second, they must all have a common mechanism for searching the knowledge dimension to identify the correct instance of a relationship with respect to the type of relationship, the values of its parameters, and the components it associates. These common mechanisms are also necessary for using the different dimensions of the model in a consistent fashion. Third, there should be a general search mechanism for finding a specific dimension in the multidimensional work environment model. These three mechanisms help establish a consistent multidimensional model that can be queried for any kind of knowledge dimension, and any work-relevant relationship therein, independent of the type of dimension and the types of relationships in the dimension. The provision of these inference mechanisms enables modeling heterogeneous relationships in the same model of the work environment to support different kinds of work, thus addressing the second part of challenge 3 in §2.4. Furthermore, such a mechanism can be used to perform a network analysis of the model of the work environment to identify dependencies between the components across different

dimensions by simply querying the model for these network dependencies (refer §4.6). To further elaborate on this, the multidimensional model can be thought of being conceptually equivalent to a multidimensional database, where one can write and run queries that fetch data elements that satisfy specific attributes on each dimension. In the case of the environment model, each database dimension corresponds to each knowledge dimension and the attributes on each database dimension correspond to the parameters that identify an instance of a relationship[1]. The query language and the multidimensional databases' internal structure establish a consistent mechanism for accessing different kind of data; similarly, the ability to query for dimensions of the work environment and the relationships in them establish a consistent mechanism for using the multidimensional model of the work environment.

3.2.3 Network Dependencies Between Workers and Work Environment

As is evident from the preceding sections, the work environment, i.e., its dimensions and its components, impose certain requirements on worker competencies. These requirements are the specification of skills, with associated internal capabilities and processors, that the worker must have to be able to successfully interact with the given work environment. For example, to execute a work process component, the worker must have a corresponding skill, and to use the functional dimension the worker must have the skill to read the dimension and find the components that are affordances to specific goals.

---

[1] The reader should note that the software and simulation architecture developed in Chapter 4 does not implement the model of a work-environment as a multidimensional database. This analogy has been drawn only for the purpose of explanation.

Thus, there are dependencies between the worker and the work environment. These dependencies can be used to construct an agent around the skills, capabilities and processors required by its work environment (further details on use of these dependencies in §4.3). Such a construction can be used to evaluate the design of the work environment based on the feasibility of equipping a worker with those skills, a form of network analysis (details in §4.6.2). For example, a work environment may require a worker to have excessive training or extraordinary abilities. Such a construction can also be combined with computational models of the skills, capabilities and processors in simulation of system behavior for operational analysis (details in §4.6.1)

3.2.4 A Rudimentary Human Performance Model

Many agent models are possible within this conceptual framework. This section describes a rudimentary human performance model that, first, adheres to the requirements and principles outlined in the preceding sections, and, second, provides a template for others to use and build on.

Workers do their work by performing activities that may achieve their goals. Thus a basic model of the worker should be able to perform activities. Figure 17 shows a rudimentary human performance model which contains an 'activity processor' that starts and performs these activities. Each of these activities is associated with the skills and capabilities that it employs, and each activity also requires a certain number of internal resources that are provided by the 'resource provider'. The example in Figure 17 has procedure following skills used for procedure following activities. For appropriate

management of these activities by the activity processor, each of them is also described by these factors:

1. The number of resources required to complete one instance of the activity. This number may be fixed or may be generated at random from a Normal distribution with a given mean and standard deviation describing the properties of the activity.
2. A relative priority, expressed as a number where higher number indicates a higher priority.
3. A time duration for each activity. During this time the activity demands the required resources, and the results of the activity may only be fully provided at the end of this time. Some activities may be continuous and thus have an infinite duration. This time can also be generated randomly based on an underlying Normal distribution to describe variability in the activity an in a human's skill.

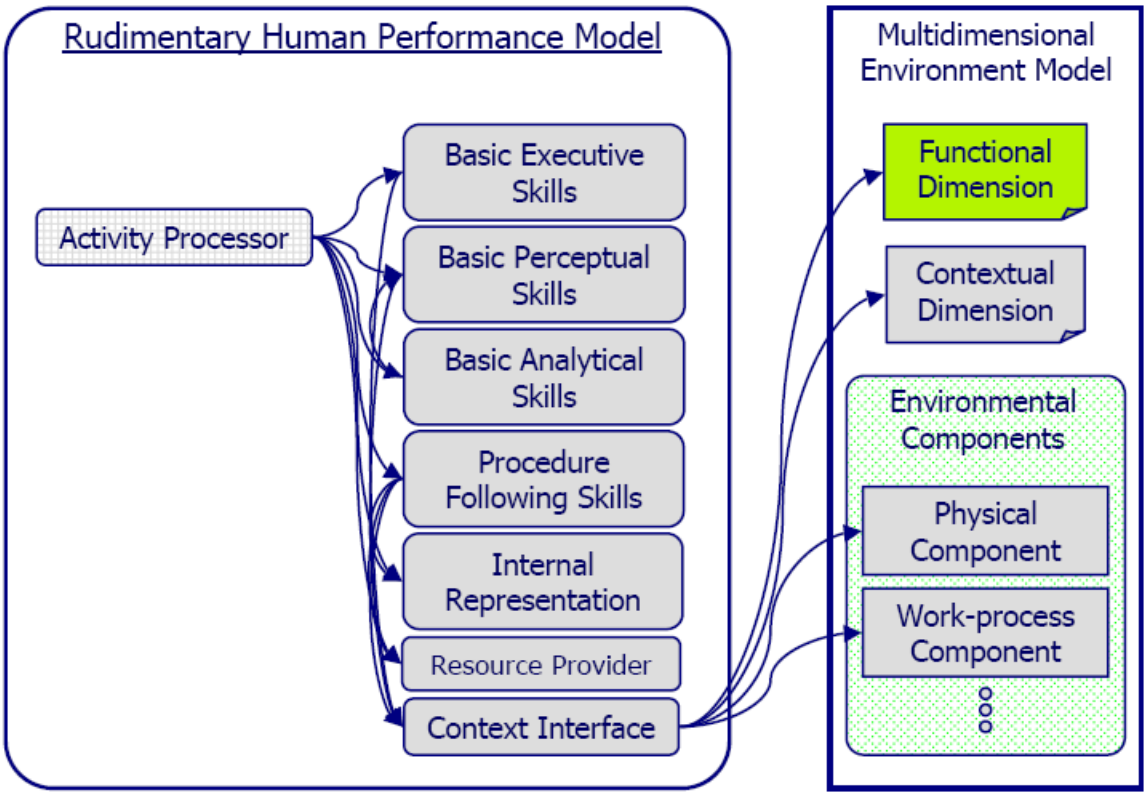

Figure 17: A rudimentary human performance model

The number of resources required at any point in time by all activities is monitored by a 'resource provider', modeled as an intrinsic capability in the agent structure. In its most basic form, the agent may have a constant amount of resources it can provide to activities. More sophisticated agent models may choose to expand the resource provider capability to allow for a changing level of available resources in response to factors such as fatigue, affect, and environmental stressors.

An 'activity processor', modeled as a processor in the agent structure, executes all activities. Specifically, when a new activity is required, the activity processor creates within itself an instance of the activity originating out of a skill or capability. The

activity processor establishes any data structures required for the activity, provides its update timing mechanisms, and effects its outcome onto the agent model and the broader environment model via its 'context interface'.

At any time the activity processor may be handling several activities. Whenever a new activity is requested, the activity processor examines the resource provider to check if resources are available to execute all current activities and this new activity. If there are, the new activity is undertaken. If there are not, the new and current activities are rank-ordered by priority. Starting from the highest priority, the activities are sequentially selected for execution (or, if already in process, are kept executing) until all available resources are used up. The remaining lower priority activities are kept in a queue of suspended activities, from which the next highest priority activities will be selected for execution as resources become available. Note that a request for a new high priority activity may 'interrupt' a currently executing activity with lower priority, i.e., bump it to the suspended queue; the data structures and time elapsed of this activity are recorded so that, when resources become available, the activity can resume where it was halted.

3.3 Summary and Discussion

This chapter developed a conceptual framework for modeling the work environment and the workers suitable for analysis of worker, component and network level transformations to the socio-technical system. This framework is inherently structure-preserving and computational, thus enabling efficient design analysis through the use of simulations. This framework identifies:

1. Which elements of the system, and their sub-elements, need to be modeled, and

2. How these elements make up the models of the work environment and the workers.

In its identification of which elements should be modeled in the work environment, the framework advances the state of the art in cognitive engineering by recognizing that the work environment consists of both physical and non-physical components. This not only makes the environment models more comprehensive, but also enables analysis of transformations enacted through these different kinds of components. In identifying how these model elements make up the work environment, the framework enhances the cognitive engineering approach by modeling the environment such that it can incorporate a number of different kinds of work-relevant relationships and is not limited to the means-ends and the parts-whole relationships that have been traditionally employed to model the work environment. This allows for analysis of different kinds of work in the system.

In this framework, the work environment is modeled on multiple dimensions, where each dimension constitutes a set of work-relevant relationships that support specific kinds of work. Each dimension is a set of the instances of the relationships between the components in the work environment; thus, changing the structure and composition within the dimension enables the designer to represent and test network level transformations. This chapter also identified and modeled two fundamental dimensions, i.e., the functional and the contextual dimensions, that are relevant to analysis of the broad range of socio-technical systems of interest to this thesis. Furthermore, each component is modeled as a collection of work-relevant internal dynamics, properties and

usage mechanisms. Changes in these elements can be used to represent component level transformations.

This chapter then discussed the requirements for the agent-based models of the workers to interact with the work environment. A worker was modeled as a collection of skills, capabilities and processors which build on each other through the use of some internal agent architecture. No specific internal agent architectures are enforced, thus allowing the designer to employ any suitable agent architecture from the broad base of architectures in agent-based modeling community; the structure of a rudimentary human performance model was provided as an example. The nature of network dependencies between worker and environment models was also established to enable network analysis and to construct computational models for simulation based operational analysis.

Figure 18 summarizes the constructs developed in this conceptual framework. The model elements in the solid blue background represent the behavioral elements of the models and those in the solid green background represent the interface elements of the components. The elements in the hashed blue background are the dimension relevant interfaces of the components that enable work-relevant interaction between components, and between components and workers.

$WE = \langle\langle C_e \rangle, \langle KD_e \rangle\rangle$	Work environment
$KD_e = \langle R_d \rangle$	Knowledge Dimensions
$R_d = \langle [C_e], \langle P_r \rangle\rangle$	Work-relevant Relationship
$C_e = \langle ID_c, \langle P_c \rangle, \langle UM_c \rangle\rangle$	Environmental Component
$C_e = \langle ID_c, \langle CA_c \rangle\rangle$	Dimension-oriented Representation of the Component
$CA_c = \langle [P_c], [UM_c] \rangle$	Aspect of a Component
$Agent = \langle\langle Skill \rangle, \langle Capability \rangle, \langle Processor \rangle\rangle$	Agent

Figure 18: Summary of the conceptual framework

Thus, this chapter conceptually tackled the challenges in §2.4. The next chapter addresses the practical challenges in instantiating the conceptual constructs developed in this chapter and simulating them.

CHAPTER 4

SOFTWARE ARCHITECTURE AND SIMULATION PLATFORM

This chapter develops a software architecture for a simulation platform and computational model structure employing the conceptual framework described in the preceding chapter. This simulation platform is design-driven to facilitate computational transformation analysis of socio-technical systems. There are primarily two kinds of challenges in creating this simulation platform:

1. Those that are general in nature and apply to the development of any agent-based simulation platform. These include:
    a. Timing and synchronization,
    b. Management of simulation runs and scenarios to assist in designing statistically analyzable experiments, and
    c. Metrics and data collection for evaluating the simulated system.
2. Those that are specific to the underlying conceptual assumptions, assertions, models and methods of the framework.

This chapter starts with a detailed description of these challenges and the requirements they generate. The features of the simulation platform that address the general simulation challenges are then described. This is followed with the description of a software architecture for computationally implementing the models developed in the conceptual

framework. Finally, the software's process is described for automatically constructing system and component models from their declarative models.

4.1 Practical Challenges for Development of the Simulation Platform

There are a number of practical challenges in developing the software architecture, i.e., the computational structures corresponding to the conceptual framework, and the simulation platform, i.e., a software application for constructing the models and simulating them. These challenges relate to both translating the conceptual framework to concrete software implementations of each agent and environmental component and to making sure that these implementations interact with each other in semantically correct ways.

When transforming the system, a designer could change the system at any or all component, worker and network levels. For instance, changes within the internal dynamics of a component or an agent may occur independently of the relationships between components. Similarly, changes at the network level may restructure the work environment, but this may not require that the internal dynamics of the components or the agents be changed. On the other hand, changes such as introducing new components into the system require changes of more than one type, including equipping workers with new skills, establishing new relations with existing environmental components, and changing the network of relationships. Since the design and transformation processes used by the designers and analysts can involve any of these scenarios, and since design is a continually evolving process (§2.3), the primary challenge is devising a modular and extensible architecture that can, first, enable structure-preserving computational models

of workers and work environment components, and, second, facilitate transformations at a combination of levels without requiring changes to existing models.

By identifying suitable constructs for modeling the system at these three levels, identifying modeling relationships between them, and preserving their structure, the framework developed in the preceding chapter has already provided a solid conceptual groundwork for such a development. What remains is the development of software architecture that computationally represents them in reusable and integrate-able software modules that preserves the structure of the system at network, worker and component levels.

Taking an object-oriented approach to constructing the computational representation helps assure that the structure of the conceptual constructs can be preserved. However, the object-oriented paradigm has no provision for semantic mappings between declarative conceptual models and their computational implementation. Furthermore, several concerns relating to the reusability of computational models of individual components are not addressed by using the object-oriented paradigm in its basic form:

1. To ensure that designers can examine a proposed system transformation without having to worry about extensive source code modifications, the software architecture must seamlessly construct computational models of the environmental components (by piecing together computational models of internal dynamics, usage mechanisms and properties), the agents (by piecing together the computational models of skills, capabilities and processors) and the overall work environment and the socio-technical system (by piecing together environmental

components and agents) while maintaining the underlying structural and behavioral semantics defined in §3.1.

2. An efficient design analysis approach requires that computational models of environmental components and workers be easily added to and removed from the overall system model, as reorganizing of such system components is a common means of transforming a socio-technical system. Likewise, a designer should be able to try different computational models representing different response behaviors for the same structural element in the declarative model of a component or agent to represent a component or worker level transformation.
3. During the course of work, due to dynamism inherent in the work environment or as a result of the activities of the workers, the structure of the work environment can change. It is important that the simulation can accommodate these dynamic changes during runtime and reflect them in the declarative models that represent the structure of the work environment, component and agent models. Whereas the preceding two issues were concerned with translating from the declarative models to computational models, this issue is concerned with the converse.
4. When dealing with real-world systems, one does not have to bother about enforcing the constraints of reality. For example, one does not have to bother about a worker not being able to see through an opaque wall or the worker not being able to press a button behind a glass cover. In simulation, on the other hand, these constraints have to be explicitly enforced, especially when the agent models can exhibit creativity. This requires a mechanism that represents

complete knowledge of the work environment in a multidimensional model yet properly constrains the worker's access to the environment.

5. If the environmental components and agents are modeled as heterogeneous systems, each needing to control their own time steps, the simulation's timing mechanism must allow for such autonomy in timing.
6. The simulation platform must satisfy some basic requirements in generating stochastic effects to ensure that the model's output has the statistical properties required for established output analysis methods, including that independent streams of random variates be used for stochastic operations across simulation runs and across components within simulation runs, and that the seed used for random number generation is controllable.
7. No two alternatives can be compared if comparison metrics have not been collected. The metrics could be collected at any level of abstraction. Thus, the simulation must provide for a common mechanism for collecting such metrics.

Whereas challenges 1 through 4 were geared at developing an architecture that can appropriately computationally represent the modeling constructs developed in this thesis, challenges 5 through 7 are specific to simulating a system. All of these concerns require integration of several advanced principles in software engineering, and from the modeling and design architectures discussed in §2.3. The next sections describe this thesis' approach to tackling these challenges, starting with those that are specific to the conceptual framework and then addressing those issues that are specific to simulating a system. §4.6 then presents this thesis approach to performing network and operational analysis.

4.2 Computational Models of the Primary Constructs

This subsection discusses how the conceptual formulations in the preceding chapter can be represented in a computational model. Computational models consist of the data-structures that appropriately represent the conceptual models, and the systematic methods or algorithms that implement these data-structures on a computer.

Broadly, this conceptual framework has model forms for two different kinds of phenomena in socio-technical systems: one model form that specifies the form of individual entities and the structure formed by relationships between those entities, and, a second that specifies the behavioral and dynamic phenomenon exhibited by those entities individually and collectively. The former is specified through declarations that appropriately categorize the entities and their relationships; the latter is specified through the use of algorithms and equations. Table 3 summarizes the mapping from conceptual constructs that may be formulated in a conceptual framework to their computational implementations. As shown, form and structure as best represented computationally through the use of declarative constructs or through specification of classes and interfaces without any specification of behavior, and behavior and dynamics are computationally represented as computer programs or numerical methods.

Table 3: Mapping between conceptual constructs and their computational implementation

Phenomena	Conceptual	Computational
Form and Structure	• Natural language or sets and tuples based specification of constructs and their associations	• Declarative models specified in languages such as XML
Behavior and dynamics	• Algorithms • Equations	• Computer implementation of algorithms • Numerical methods for solving equations • Specification of interaction protocols between classes

As noted in the preceding chapter's description of the conceptual framework, the declarative models describing form and structure are expressed in the form of nested sets and tuples that are searchable based on a partial or complete specification of their attributes. Thus, the computational formulation of the structure of these models is merely a specification of symbols, value attributes, semantics and search indices in a data representation language tailored to the structure of the conceptual models. These models can thus be represented declaratively with the use of any relationally complete data representation and query language (Codd, 1970). Since relational algebra, tuple calculus and set theory are mature topics, for a detailed description of these models, including their logical correctness and appropriateness for computational analysis, the reader is referred to (Cantone, Omodeo et al., 2001).

However, such declarative specifications are only suited for network analysis of the work environment and its components, i.e., analysis of relational dependencies between the components and their constituent model elements. By themselves they are not sufficient

for operational analysis in terms of the behavior of individual components and emergent system performance. To enable such operational analysis, the internal dynamics of the components have to be modeled computationally. Since the components of the work environment may be heterogeneous, and the algorithms and data structures that model their internal dynamics may vary in their level of complexity and the nature of their constructs, it makes no practical sense to devise a homogeneous computational model in terms of a symbolic language capable of representing all kinds of processes (linear and non-linear, continuous, discrete or hybrid) governing the internal dynamics of all kinds of components and agents in the system. This thesis has therefore chosen to use an object-oriented design methodology to encapsulate the internal dynamics of the components and to abstract their complexity away from the models that are meant to model and analyze relational dependencies at the component and network levels. The following subsection discusses the declarative formulation and §4.2.2 details these object-oriented models with the software architecture.

4.2.1 Declarative Models

Declarative models semantically represent the relationships between the model elements to specify the structure of and the interrelations between the components of the system. Declarative models serve two purposes: 1) they allow composition of complex component and system models by specification of the model constituents and structural relationships between them, and 2) they enable network analysis through use of structured representations and querying for inference making.

Due to the availability of a number of off-the-shelf tools for working with the declarative modeling language XML (eXtensible Markup Language), this thesis has chosen to use it to computationally specify the structure of the models of the agents, the components and their network (Bray, Paoli et al., 2004). XML represents any structural model using a tree data structure, the constraints on the structure and syntax of which can be formally specified using a Data Type Definition language or an XML Schema. This structure and syntax can be related to any semantics specific to a domain, thus helping formulate a domain specific language with its own established vocabulary and grammar. Thus, when using XML to define a declarative modeling specification for any domain, there are two primary tasks: defining the structure of the document that holds the instances of the models, and defining a vocabulary and grammar that establishes the model semantics and associating it with the document structure. In the preceding chapter this thesis has already established the conceptual semantics of the constructs; the following subsections define the structure of the documents that syntactically and structurally represent the modeling constructs.

4.2.1.1 Declarative Specification of a Component

The XML specification of the radar equipment for sector ZLA-39 is shown in Figure 19. In keeping with the specification of a component in Model 5 (§3.1.1), this component is built through assembling internal dynamics (`WEAInternalDynamics`) and aspects (`WEAAspects`). The single aspect in this example lists one property and two usage mechanisms. This aspect provides the contextual dimension relevant interface of the radar equipment, analogous to having a radar screen (also refer Figure 13 in §3.1.1). This XML specification also lists an internal dynamics element of the radar that represents its

airspace scanning behavior. This specification also shows that each model element, i.e., the aspect, internal dynamics and the component, has an `initialization` element that is used both to initialize the computational instance of that model element and to link up the model elements to construct the models of the components, dimensions and the complete work environment. The contents of the `initialization` element are populated both to set the initial values for parameters within the component, and to specify interrelations between model elements. For example, the properties and usage mechanisms are both declared and associated with the underlying object-oriented implementation (through the specification of 'ODMExxx' binding) in the initialization element. Similarly, the internal dynamics is made aware of the interface elements by linking it to the '`RadarData`' aspect. When creating a component model any number of aspects can be added onto the component, and each can declare any number of properties and usage mechanisms that reference specific elements in the component's internal dynamics. These aspects can overlap on the properties and usage mechanisms they expose. The construction of the component model can be changed simply by adding, removing or replacing these aspects, or the properties and usage mechanisms they declare. The model can also be changed by adding, removing or replacing any internal dynamics element of the model with another element that provides the same function to the component's properties and usage mechanisms, but changes the behavior internal to the component. The underlying object-oriented implementation is identified through the DLL (Dynamic Link Library) attribute and the ClassID of the implementation in that specific DLL (a detailed discussion of this dynamic linking is given in §4.3).

```
<WEAComponent Name="Radar39" ClassID="WEAComponent"
      DLL=".\modules\WEA\RFS_WEABase.dll">
  <Initialization><![CDATA[
    <Dimension type="ContextualDimension">
      <ContextualAspect Name="RadarData"/>
    </Dimension>
  ]]></Initialization>

  <WEAAspects>
    <WEAAspect Name="RadarData" ClassID="WEAContextualAspect"
        DLL=".\modules\WEA\WEAContextualAspect.dll">
      <Initialization><![CDATA[
        <ParentContext Name="//ZLA39"/>
        <WEAProperties>
          <WEAProperty Name="ACListXML" ODMEVariableName="ACListXML"/>
        </WEAProperties>
        <WEAUsageMechanisms>
          <WEAUsageMechanism Name="GetACCoordinates" ODMEMethodName="
              getACCoordinates"/>
          <WEAUsageMechanism Name="GetACSpeed" ODMEMethodName="
              getACSpeed"/>
        </WEAUsageMechanisms>
      ]]></Initialization>
    </WEAAspect>
  </WEAAspects>

  <WEAInternalDynamics>
    <WEAProcessor Name="ScanAirspace" ClassID="ScanAirspace"
        DLL=".\modules\WEA\WEARadar.dll">
      <Initialization><![CDATA[
        <RadarDataPrefix Value="RadarData"/>
        <SectorName Value="39"/>
        <RadarDataAspect Value="RadarData"/>
      ]]></Initialization>
    </WEAProcessor>
  </WEAInternalDynamics>
</WEAComponent>
```

Figure 19: Declarative specification (XML) of a radar equipment

4.2.1.2 Declarative Specification of an Agent

Declaratively, an agent is specified in much the same way as a component, but the composing elements are different, both in their XML element names and in their underlying semantics. Instead of the aspects (`WEAAspects`) and internal dynamics (`WEAInternalDynamics`) shown in Figure 19, an agent has skills (`WEASkills`), capabilities (`WEACapabilities`) and processors (`WEAProcessors`) (refer to the example in Figure 20). Initialization of these elements is used both to set the initial values of parameters in each of these composing elements, and to establish interrelations between them that establish the internal architecture of the agent. For example, the resource provider in Figure 20 is initialized to have a maximum of seven resources and the activity processor is initialized to reference this resource provider when generating activities.

```
<WEAAgent Name="ATC39" ClassID="WEAAgent"
      DLL=".\modules\WEA\RFS_WEABase.dll">

  <WEASkills>
    <WEASkill Name="ProcedureFollowing"
        ClassID="WEAProcedureFollowingSkill"
        DLL=".\modules\WEA\WEAProcedure.dll">
      <Initialization><![CDATA[
        <DefaultContextAccessor Value="SectorContext"/>
      ]]></Initialization>
    </WEASkill>
    … …
  </WEASkills>

  <WEACapabilities>
    <WEACapability Name="ATCResourceProvider"
        ClassID="HumanResourceProvider"
        DLL=".\modules\WEA\WEAATC_R.dll">
      <Initialization><![CDATA[
        <MaximumResources Value="7"/>
      ]]></Initialization>
    </WEACapability>
    … …
  </WEACapabilities>

  <WEAProcessors>
    <WEAProcessor Name="ActivityProcessor"
        ClassID="WEAActivityProcessor"
        DLL=".\modules\WEA\WEAActivityProcessor.dll">
      <Initialization><![CDATA[
        <ResourceProvider Value="ATCResourceProvider"/>
      ]]></Initialization>
    </WEAProcessor>
    … …
  </WEAProcessors>
</WEAAgent>
```

Figure 20: Partial declarative specification (XML) of an air traffic controller agent

4.2.1.3 Declarative Specification of the System

The declarative specification of the entire system first lists all components and agents with the timestamps of when they appear in the system (Figure 21). This creates a time-based collection of all components in the system. The following subsections illustrate

how work-relevant relationships are then established between these components to structure the work environment in one or more dimensions.

```
<WEATimedScript xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
    xsi:noNamespaceSchemaLocation="RFS_WEATimedScript.xsd">
  <TimedScript Time="0">
    <LoadComponents>
      <WEAComponent Name="RandomNumberGenerator"
          ClassID="WEARandomNumberGenerator"
          DLL=".\modules\WEA\WEARandomNumberGenerator.dll">
        … …
      </WEAComponent>
      … …
    </LoadComponents>
  </TimedScript>

  <TimedScript Time="100">
    <LoadComponents>
      <WEAAgent Name="ATC39"
          DLL=".\modules\WEA\RFS_WEABase.dll"
          ClassID="WEAAgent">
        … …
      </WEAAgent>
      … …
    </LoadComponents>
  </TimedScript>

  … …
</TimedScripts>
```

Figure 21: Declarative specification (XML) of a system model

4.2.1.4 Declaratively Constructing the Dimensions

As discussed in §3.1.1, work-relevant relationships are established through mapping the components onto the dimensions and defining the values of the relationship parameters.

Figure 22 and Figure 19 together illustrate how the contextual dimension is constructed. The contextual aspect in Figure 19 lists an element named '`ParentContext`' that establishes the contextual-composition relationship with a parent contextual node within the existing structure of the contextual dimension. The value of this element identifies the parent contextual node; for example, the '`RadarData`' aspect in Figure 19 identifies the contextual node of the sector ZLA-39 as its parent contextual node, thus putting it in the workspace of the controller of sector ZLA-39. But, before this relationship can be established, the ZLA-39 contextual node should exist. Figure 22 shows the declarative specification of the component ZLA-39 that represents that sector. This component has a contextual aspect that maps this component onto the contextual dimension as a top-level contextual node, i.e., a contextual node that subsumes other contextual nodes but is itself not subsumed by others. This component and its aspect have to be created in the system before the radar is created, so that when creating the radar the relevant relationship can be established.

It should also be noted that each component lists its mapping to specific dimensions in initialization (Figure 22 and Figure 19). This listing identifies which aspect in the component maps onto each dimension. The aspect's initialization then lists how exactly the relationship is established. Figure 23, the declarative model representation of the 'Heading Merge' work-process, shown conceptually in §5.2.1 in Figure 36, exemplifies how a component can be mapped onto more than one dimension, by listing those dimensions and providing the respective aspects on the component.

```
<WEAComponent Name="ZLA39" ClassID="WEAComponent"
    DLL=".\modules\WEA\WEAComponent.dll">
  <Initialization><![CDATA[
    <Dimension type="contextualDimension">
      <ContextualAspect Name="ZLA39"/>
    </Dimension>
  ]]></Initialization>

  <WEAAspects>
    <WEAAspect Name="ZLA39" ClassID="WEAContextualAspect"
        DLL= ".\modules\WEA\WEAContextualAspect.dll">
      <Initialization><![CDATA[
        <ParentContext Name="//"/>
      ]]></Initialization>
    </WEAAspect>
  </WEAAspects>
</WEAComponent>
```

Figure 22: Declarative specification (XML) of sector ZLA-39

```
<WEAComponent Name="HeadingMerge" ClassID="WEAWorkProcess"
      DLL=".\modules\WEA\WEAWorkProcess.dll">
  <Initialization>
    <Dimension type="FunctionalDimension">
      <FunctionalAspect Name="Func"/>
    </Dimension>

    <Dimension type="ContextualDimension">
      <ContextualAspect Name="HeadingMerge"/>
    </Dimension>

    <Procedure>
      … …
      … …
    </Procedure>
  </Initialization>

  <WEAAspects>
    <WEAAspect Name="Func" ClassID="WEAFunctionalAspect"
        DLL=".\modules\WEA\WEAFunctionalAspect.dll">
      <Initialization><![CDATA[
        <MCER MCF="means" CAT="WorkProcess" Goal="AvoidConflict"/>
      ]]></Initialization>
    </WEAAspect>

    <WEAAspect Name="HeadingMerge" ClassID="WEAContextualAspect"
        DLL=".\modules\WEA\WEAContextualAspect.dll">
      <Initialization><![CDATA[
        <ParentContext Name="//ZLA39"/>
        <WEAProperties>
          <WEAProperty Name="Situation"
              ODMEVariableName="SituationXML"/>
          <WEAProperty Name="Process" ODMEVariableName="ProcessXML"/>
        </WEAProperties>
      ]]></Initialization>
    </WEAAspect>
  </WEAAspects>
</WEAComponent>
```

Figure 23: Declarative specification (XML) of a work-process component

As the system model-constructing facility (refer §4.3) reads through the specification of each component in the system model, it automatically constructs the contextual dimension, illustrated in Figure 24 for the example of the radar equipment, where the automatically constructed specification of the contextual dimension represents a hierarchical structure of the controller workspace. Through the use of such a component-oriented construction of the model of the work environment, the system designer can easily model a network-level change by simply adding or removing a component from XML representation of the system, or by making a change in any of its aspects' XML specification to change its relationships with other components. Since the models of the dimensions are constructed automatically, network analysis can be performed at any point in time (described in more detail in §4.6.2).

```
<ContextualDimension>
  <ContexutalNode Name="ZLA39" WEAComponent="ZLA39"
      WEAAspect="ZLA39">
    <ContextualNode Name="RadarData" WEAComponent="Radar39"
        WEAAspect="RadarData">
      <Properties>
        <Property Name="ACListXML"/>
      </Properties>
      <UsageMechanisms>
        <UsageMechanism Name="GetACCoordinates"/>
        <UsageMechanism Name="GetACSpeed"/>
      </UsageMechanisms>
    </ContextualNode>
  </ContextualNode>
</ContextualDimension>
```

Figure 24: Automatically constructed declarative specification (XML) of the contextual dimension

4.2.2 Object-Oriented Models

This section starts with a discussion of two foundational computational constructs that have been developed to facilitate building the models of the environmental components and the agents such that system-level models can be seamlessly assembled from them and used in agent-based simulations. These constructs are *objects* and *facets*. These fundamental constructs incorporate a number of architectural features that are key to solving most of the fundamental challenges discussed in §4.1.

This section then describes constructing the computational models of the environmental components and the agents using these constructs. A computational model of the knowledge dimensions is presented. Specific attention is paid to translating from computational models to declarative models that specify the interrelations between components. The computational model of the work environment is then described.

4.2.2.1 Objects and Facets

Objects and facets are the foundational computational constructs in this software architecture. These constructs were introduced primarily to tackle challenges 1 and 2 from §4.1, i.e., to seamlessly construct the computational models of the environmental components, the agents and the system, and to provide for interchangability of computational implementations. The structural properties of these constructs maintain the underlying semantics of the conceptual framework.

These constructs have been implemented as object-oriented classes. The object class is the base class for any entity modeled in the system; the environmental component class and the agent class derive from this class. An object class is simply a container of facets

(Figure 25), where a facet is a basic unit of constructing any functionality in the object (including both type of phenomena shown earlier in Table 3, i.e., form and structure and behavior and dynamics). The object aggregates all elements of all the facets that it contains (Figure 26).

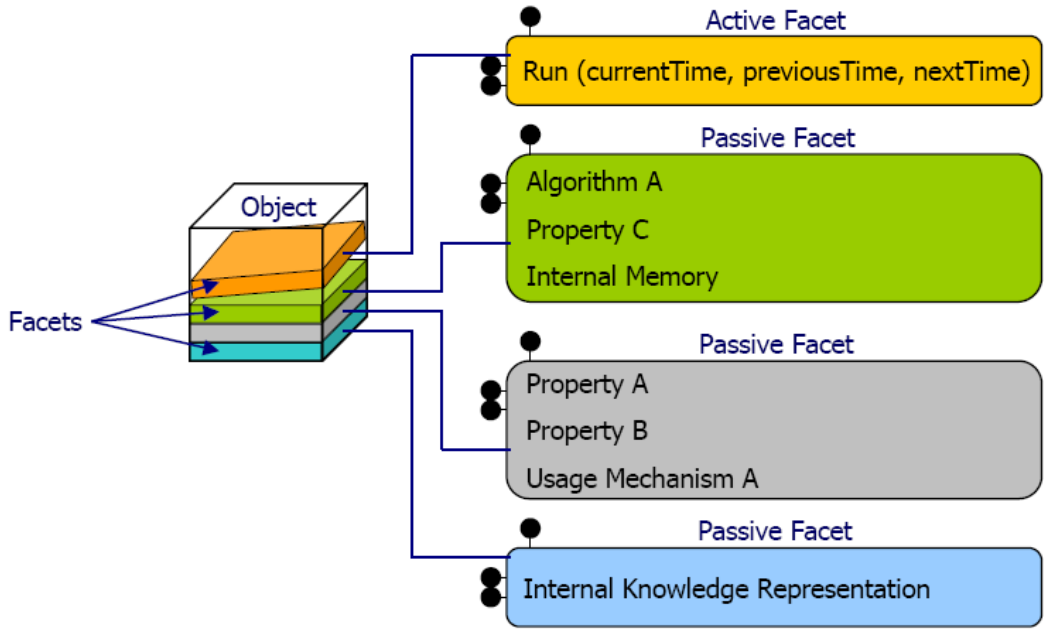

Figure 25: Example of object and its facets

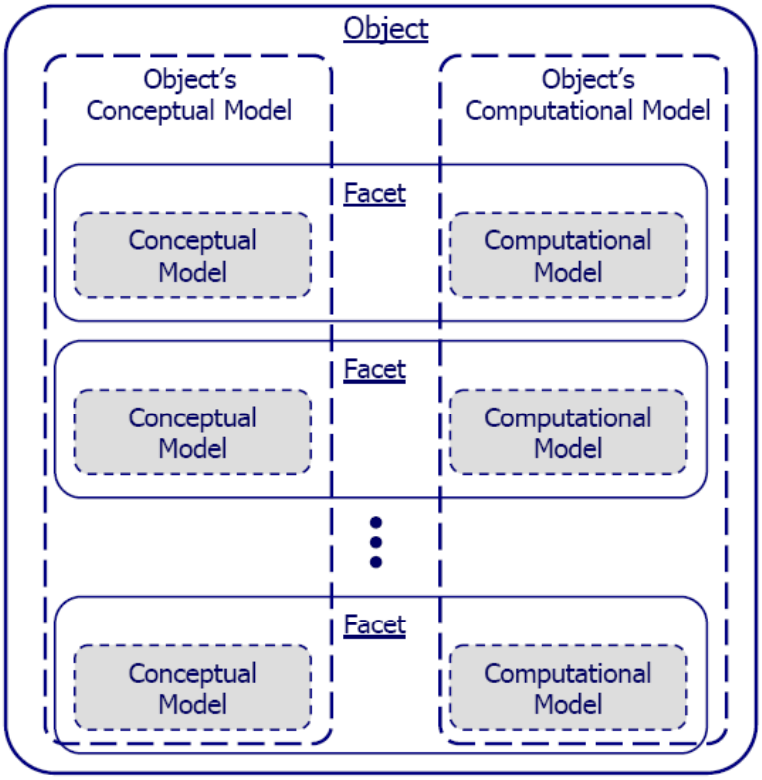

Figure 26: Aggregating facets to create object models

There are two primary kinds of facets: passive and active. An active facet represents autonomous dynamics. Such a facet is used in agent models to represent their autonomous processors (§3.2) and in environmental component models to represent their self-updating internal dynamics (model (1) in §3.1) and thus many include both computational implementation of equations and algorithms to describe temporal dynamics and declarative descriptions of the component's form and structure. Each active facet (Figure 25) includes the *Run* method that is invoked repetitively by the timing mechanism of the simulation platform to simulate the internal dynamics of the object. This method is provided with the current simulation time, the previous update

time, and a parameter to return the next time the active facet wants itself to be updated, thus providing it the capability to govern the accuracy of the internal dynamics that it computationally models (§4.5.1).

Passive facets provide the object with computational storage and algorithms both for internal use and for use by other objects. As with active facets, passive facets can describe both type on phenomena noted in Table 3. Some storage and algorithms may be exposed as properties and/or usage mechanisms to other components in the system (Table 2).

A facet can be dynamically added onto an object or taken away from it. By virtue of the object accumulating both the declarative and equation/algorithm elements of the facets that it aggregates (Figure 26), a facet is the basic unit of conceptually and computationally constructing environmental components and agents, and their internal elements such as aspects and skills. Active and passive facets can be specialized to represent both the internal dynamics of environmental components and the aspects that map the component onto the dimensions of the work environment. A facet can also be specialized to represent a skill, capability or processor of an agent.

Given the flexibility of this architecture, it is always possible to abuse these constructs and make big monolithic models of the complete agent in one facet. Such a practice would completely defeat the purpose of developing these constructs, for one should always maintain close correspondence of these constructs with the basic and most fundamental units of constructing models of a system and its components within the semantics of the conceptual framework of this thesis.

Computationally, a facet is an object-oriented class that encapsulates data and functionality. Unlike an ordinary C++ class, this class has some added infrastructure that allows it to be dynamically deployed and aggregated into the computational model of any object even after it has been compiled into native machine code, i.e., it does not require source code level integration for aggregating its functionality into the object. This infrastructure includes declaring some of its C++ data as component properties, and some C++ methods as component usage mechanisms. These declarations expose the conceptual model that the aspect computationally represents (Figure 26). Likewise, these declarations are used by the work environment and agent model construction architecture (§4.3) to translate between the conceptual declarative model and its computational representation. This infrastructure builds on mechanisms for in-process dynamic linking (refer to client-server software architectures in (Rogerson, 1997)) that are meant to enable runtime loading and functional linking of the precompiled C++ code. A second feature of this infrastructure is the principle of aggregating a facet into the object (Figure 26) in such a way that any other object working with the aggregated object does not need to know about the internal structure of the object. Computationally this feature builds upon a combination of two principles known as *aggregation* and *automation* that have been adopted from the field of Component Based Software Engineering (Brown, 1996; Heineman and Councill, 2001; Rogerson, 1997).

By encapsulating this infrastructure in these two foundational computational constructs, and extending them to build the computational representation of the models developed in the conceptual framework, this thesis addresses the practical challenges 1 and 2 in §4.1. These constructs provide the mechanism to assemble the component, agent and system

models from disparate model elements, while also leaving the room for maintaining their conceptual semantics. The next few sections discuss those models based on these two constructs.

4.2.2.2 The Environmental Component

In model (1) in §3.1, an environmental component was modeled as a collection of internal dynamics, work-relevant properties and work-relevant usage mechanisms. Internal dynamics can include both autonomous processes and responses to actions. These dynamics are computationally represented through algorithms that use state variables from within and outside the component. Algorithms that can command their own temporal characteristics without external requests represent the autonomous internal dynamics of the component, whereas algorithms commanded only by external requests represent the component's passive responses. In this software architecture, such algorithms are packaged in active and passive facets that can be aggregated into the computational model of the environmental component (§4.2.2.1). Algorithms for autonomous processes are packaged in active facets and must implement the *Run* method (§4.2.2.1), while responses are packaged in passive facets.

Component properties are modeled as state variables, the state of which is changed by the internal dynamics of the environmental component. A C++ class implementing a facet can have any number of such variables that may be publicly exposed to other classes, but those variables that represent properties in the declarative model of the environmental component (Model (1) in §3.1) are declared through a special mechanism that enables them to be accessed without any C++ level knowledge of their implementation, i.e., with

the use of only knowledge about the component model represented in its declarative model as illustrated in §4.2.1.

Usage mechanisms are computationally modeled as C++ methods. These methods can be invoked by models of workers as means to use the environmental component, i.e., take actions on it. For example, a toggle switch on a switchboard may be conceptually represented as a usage mechanism for a worker. The computational algorithm that implements the toggle switch would invoke the internal dynamics of the component that represents the component's response to such usage. Similar to the properties, they are also declared using a special mechanism that enables them to be accessed without any C++ level knowledge of their implementation and to be semantically associated with the declarative model. Similar to the properties, they are also encapsulated in facets that can be aggregated into an object.

To represent a component as a collection of aspects (Model 5 and 5a in Chapter 3), each aspect should be computationally represented as a facet, and mapping them onto a given environmental dimension. One should note that it is not required that the facet representing the aspect computationally implement all the properties and usage mechanisms as per model (5); it can also reference the implementation of other facets thus making it possible to have overlapping aspects representing overlapping sets of properties and usage mechanisms on distinct dimensions of the work environment model. Whereas it is not required that internal dynamics and aspects maintain one to one mapping with the facet that implements them (each of them can be implemented by many, thus providing for an additional level of conceptual granularity), system modelers

may choose to do so as a best practice to maintain clear correspondence between declarative and computational models.

A component model is constructed by assembling all the facets that represent its internal dynamics, and its aspects (and thus its properties and usage mechanisms). The model construction architecture discussed in §4.3 processes the XML representation of the list of the constituent facets, and constructs the component model by assembling the computational representation of the facets into an empty object. The internal linkages amongst the facets are also established to represent any internal architecture of the component model. Furthermore, it uses the property and usage mechanism declaration mechanisms to construct the declarative model of the environmental component.

While the XML listing of facets is used to construct the 'set' type model of the environmental component as described in Model (1) and (5) in §3.1, there are other features of this computational model that foster reusability and flexibility in constructing complex internal architectures for a component. The need for this is exemplified in the toggle switch discussed above, in which the toggle switch invokes internal dynamics of a component. Since the computational model of the toggle switch usage mechanism could possibly be represented in a facet different from the facet that models the internal dynamics, the toggle switch has to be able to invoke the computational model of the internal dynamics without knowing about it at a C++ level. This is where aggregation of conceptual models of facets comes into use. The toggle switch facet has to know only about declarative existence of the internal dynamics in the object; with that it can computationally invoke the internal dynamics through the object itself, without having to concern itself with which facet provides that functionality. Thus, a complex component

can be built, the internal dynamics and usage mechanisms of which build up on the functionality provided by different facets in the same object, without having to know about their C++ classes.

4.2.2.3 The Agent

§3.2 discussed the conceptual model of a worker that is a collection of skills, capabilities and processors. Computationally these elements are modeled in the same way as the environmental components' internal dynamics, i.e., through instantiating them as object-oriented classes and encapsulating them as facets that are aggregated into the agent modeled as an object.

This simple mechanism can account for complex internal agent architectures. Figure 27 shows an example of an agent architecture built using simple facets. In Figure 27 facets are identified as shaded round edged rectangles. All the facets that construct the specific agent model are enclosed in the dotted rectangle in the agent model. In the figure, everything outside the dotted rectangle is modeled inside a generic computational model of an agent, which is a specialization of the generic object discussed in §4.2.2.1. All the facets on the right side of the dotted section are passive facets and account for internal resources such as memory, and internal representations of visual and auditory perceptions. Each of these resources has its own way of storing information that is coded by the designer of these facets using any kind of data structure and algorithms they deem appropriate. These models are coded in C++. Since there is no limit on the number of facets that can be aggregated into an agent model and on what they model, in principle one can build on underpinnings of the cognitive sciences and artificial intelligence to

construct sophisticated agent models including all of the capabilities suggested by Figure 27.

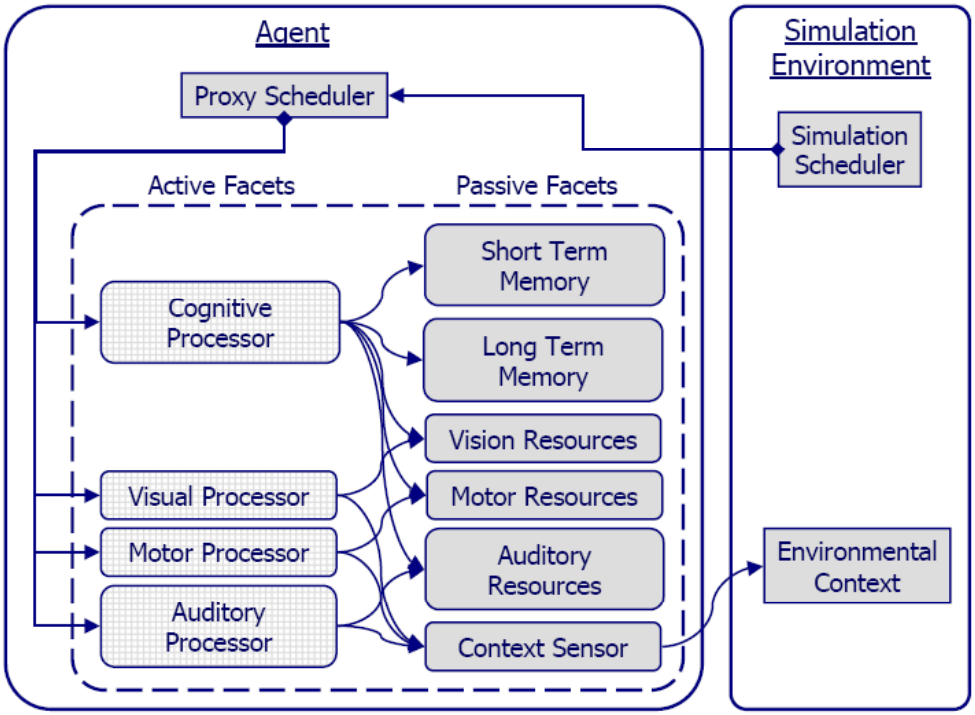

Figure 27: Example of agent model constructed from facets

The left hand side of the dotted rectangle has the agent's active facets that are essentially internal processors that can operate in parallel. The proxy scheduler within the agent model runs each of these processors asynchronously within the agent, where each processor can command its own update time step (§4.5.1). Thus, the agent model is capable of incorporating very complex parallel and autonomous processing models,

which can, in principle, command their update rates to ensure any level of accuracy in its model. These processors can interact with other facets in the model, i.e., the facets that represent internal resources and facets that enable an agent to sense and act upon its environmental context, to process their data and to change internal state and external behavior of the agent.

Ordinarily, when making an object-oriented model, the interactions between facets would be expressed directly as shown in Figure 27. However, this would mean that; when constructing the computational model of the agent, the developers of one facet would have to know about the computational implementation of each of the other facets that it interacts with. This would not allow one facet of an agent model to be modified or replaced without concomitantly changing the rest of the model (contrary to the requirements listed in §4.1). To remove such a limitation, the mechanism for conceptual model aggregation afforded by the objects and facets constructs comes in handy. To access the functionality of other facets a facet does not have to know about the code of these facets or even how these facets work with each other. Using the conceptual model aggregator, each passive or active facet directly interacts with its object and the conceptual model aggregator of the object directs their interaction to the correct facet (Figure 28, Figure 26). The conceptual model aggregator uses the knowledge of skills, capabilities and processors represented in the declarative model and brought in by each facet to direct this internal interaction between facets.

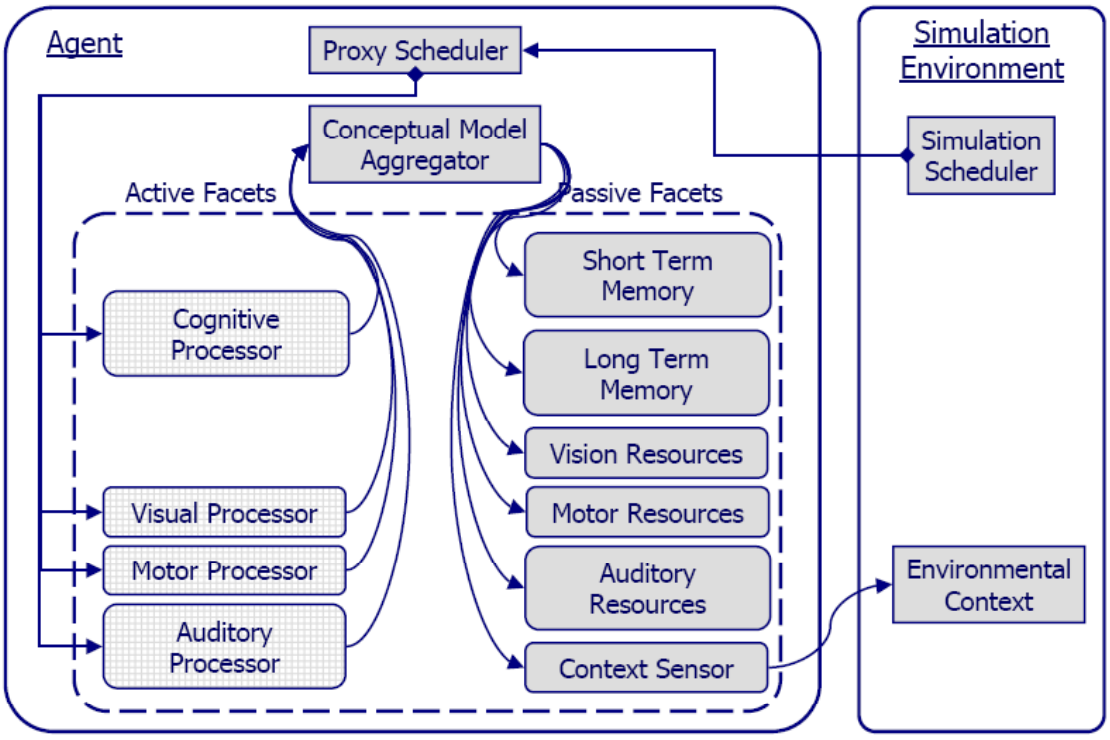

Figure 28: Using the conceptual model aggregator to access facets

4.2.2.4 Environmental Knowledge Dimensions

The previous subsections discussed the computational models of the environmental components and the agents. These models had internal dynamics that computationally processed internal data and data from their environmental context. A dimension, on the other hand, does not have any internal processing, but is instead a structured declarative representation of relationships between the components of the work environment. Nevertheless, the computational models of the worker need to use it. Likewise, the structure of the dimension may also dynamically change during the simulation.

As described by Model (4) in §3.1, a dimension is a collection of declaratively specified relationships that associate sets of environmental components and have certain relationship-defining parameters associated with each of them. To use a dimension a worker requires a general search mechanism that searches for relationships in the dimension based on certain value attributes.

Based on such characteristics of usage and the nature of represented work-relevant relationships, this thesis uses existing mechanisms of declarative data representation, retrieval and storage using the Document Object Model (DOM) (Hors, Hégaret et al., 2004). The document object model is a World Wide Web specification for representing structural data in a document with a tree-like structure. Any author can define their own vocabulary and grammar (semantics) for the content of the document using XML Schema (Fallside and Walmsley, 2004). Others can use that schema to create a XML document (Bray, Paoli et al., 2004) based on the underlying language specification. Users of the document can use a validating parser such as the MSXML (http://msdn.microsoft.com/XML/) that validates the document for a given vocabulary and grammar. This parsed document can be queried using the XPath and XQuery specifications (Boag, Chamberlin et al., 2005; Clark and DeRose, 1999) for the specifics of the data they want to obtain. The same DOM used to read the parsed XML can also be used to write back to the document even while reading from it, thus providing the ability to update it while adhering to its vocabulary and grammar.

This general mechanism was adopted to represent the dimensions as XML documents (refer to §4.2.1 for the structure of those documents). For each new dimension, an XML schema was established which included the definition of each work-relevant relationship

included in the dimension and the structure of the document that lists those relationships. Each relationship is defined in terms of its associated components, any restrictions on the nature of those components, and the number of those components. Building up on model (3) and (4), each relationship also defines parameters whose values define each instance of the relationship. When creating a domain specific model, the dimensions of a specific work environment are each represented in XML documents, each based on the XML Schema associated with a specific type of dimension.

For the workers to be able to use a dimension, they should be equipped with a specific skill or capability that can parse the XML representation of the dimension and then use it. For example, the contextual dimension lists the properties and usage mechanisms in a particular contextual node. A worker skill for reading the contextual dimension would be able to find an invoke usage mechanisms.

If the model of the work environment changes during the simulations in terms of the added or removed components or changed relationships, the dimension is updated using the DOM, thus tackling challenge 3 in §4.1, i.e., ensuring that the declarative model of the system matches the current state of the computational model.

4.2.2.5 The Work Environment

As discussed in Model (2) in §3.1, the model of the work environment includes a collection of all the components in the work environment and all the dimensions on which the work environment is modeled. This is probably the simplest of all computational models implemented in this thesis. It simply has two searchable collections that accumulate all the objects and all the dimensions in the environment

model. Every worker model in the simulated system has a reference to this environment model to enable it to access the dimension it wants to refer to.

4.3 Constructing the System Computational Model

This section discusses how the declarative model of the system, constructed by piecing together the declarative models of its components, is translated into the computational model of the system. This computational model is an assembly of the computational models of the components and their internal elements, and can be simulated for operational analysis. As discussed in §2.3, socio-technical systems are designed by piecing together system components such as technology, processes and information through specific work-relevant relationships. When creating a system model using structure-preserving models of these system components, the system model is pieced together in the same fashion as the real world. In imitation of reality, when constructing the model a designer might go through several iterations, incrementally changing the designs by changing the relationships between the components or by adding and subtracting components. In doing so, the designer does not want to restart with every single piece that was used before, but only work with the changes. This thesis' structure-preserving models are meant to enable this.

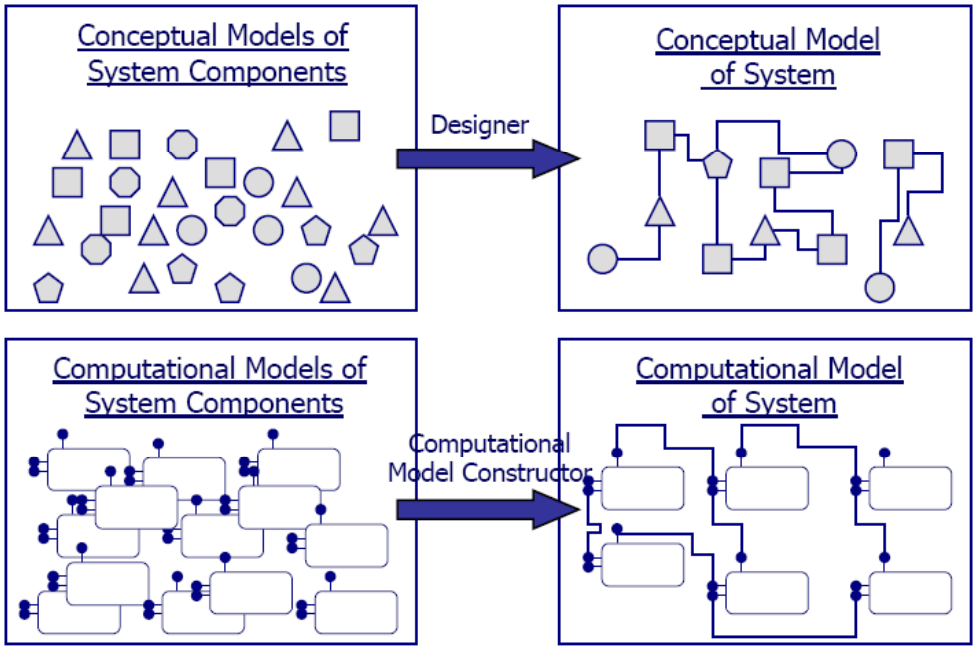

Figure 29: Correspondence between conceptual and computational models of the system

Learning from the principles of component based software engineering and model driven and design driven architectures (refer §2.3), this thesis devised special mechanisms in its computational modeling constructs that seamlessly create a computational equivalent of the system conceptual model (Figure 29) without the designer having to worry about how the implementations of the classes will come to know about which C++ methods they should call in a potentially large and unfamiliar software development.

As mentioned in the section on objects and facets, there are certain features of the facets that enable them to declare certain C++ elements such as variables and methods relating to declarative elements of the model. Through such declarations, a facet essentially has two parts to it: a conceptual part and a computational part. The conceptual parts are associated with globally unique identifiers that are specified in the form of a string of

characters. These globally unique identifiers are separated from the code and associated with more human-readable identifiers that the designer uses as the conceptual equivalent of the modeling constructs when constructing the XML based declarative models. The designer can create the complete declarative model of the system through the use of these identifiers. Once the declarative model has been created, its XML representation is fed to the computational model constructor that pulls out the globally unique identifiers from a registry that stores a map of the conceptual model's identifiers with the code identifiers (Figure 30). It then picks up the computational model (i.e., the C++ implementation packaged in a Dynamic Link Library (DLL)) associated with each of them and assembles the system model (Figure 30). This is the first phase of this translation, which is only related to mapping from the declarative models of the components to the computational models.

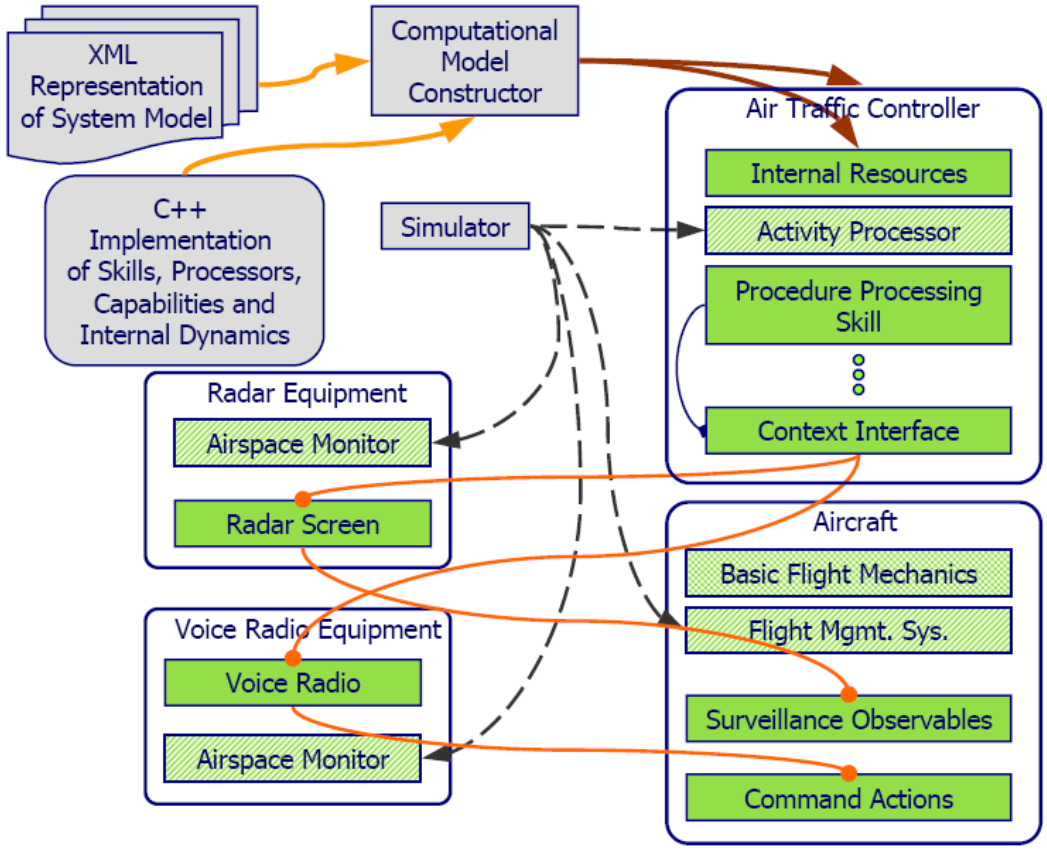

Figure 30: System model generation architecture

The second and the more detailed phase is the use of in-process dynamic linking. It may be remembered from §4.2.2.1 that facets expose their conceptual functionality by exporting their variables and functions outside of their code space in the computer's memory, thus making them available for runtime use by other computational models without the user classes having to know about the source code of the facet. The computational models interact with each other through the use of these exported functions using their conceptual model, i.e., via declarative handles, thus making it possible for the precompiled code to be used seamlessly while maintaining complete

coherency with the declarative models. This is how the architecture tackles the first challenge in §4.1, i.e., seamlessly constructs the computational model of the environmental components, agents and the system from their declarative representations.

Challenge 2 in §4.1 mentioned a scenario in which the designer or system analyst may want to maintain the network level model of the system and the declarative model of the components but change the underlying implementation of dynamics of the components, i.e., a component level transformation. This challenge is tackled with a slight enhancement of the above scheme in which a human friendly identifier of a conceptual model was associated with a globally unique identifier that associated itself with the code base. To accommodate challenge 2, this mapping is enhanced to link more than one globally unique identifier with each conceptual modeling construct. In this case all the designer has to do is pick the version of the computational model he or she wants to use by identifying the globally unique identifiers of the required version of code for the same conceptual construct. The computational model constructor then references that version of the computational model, which provides the same conceptual interface of the model. The different versions of code, when grouped together to represent a group of conceptual constructs for a class of agents or components, are collectively referred to as a template. Instead of choosing a globally unique identifier for each conceptual construct the designer can simply choose a template to represent a change in underlying computational models as a whole group.

While the theme of the above discussion was centered on the model of the system, the same principle is used at the lower level of abstraction when constructing the computational models of the environmental components and the agents.

As an additional development, agent models can be generated automatically from analysis of the work environment when some assumptions can be made about the workers in the system relating to their intrinsic skills and capabilities. Similar to CWA, this thesis assumes a distribution of goals amongst the workers in the transformed system, from which all environmental components that may afford or constrain the goals can be identified. Second, the worker is assumed to access only those environmental components that are in their context. A third assumption is that the worker is aware of certain dimensions of the work environment (CWA assumes the worker is aware of two dimensions, while this thesis' framework does not restrict which dimensions are considered). With these three assumptions, the worker-relevant components and the dimensions are identified. A model of a worker or a class of workers is then established with all capabilities and skills that are needed to a) work with the chosen dimensions, and b) to interact with the environmental components through their properties and usage mechanisms that map to the chosen dimensions. The model of the worker can also be equipped with some intrinsic skills and capabilities to which this ideal set of capabilities and skills is added. This process is termed in this thesis as "work environment centered agent generation" or simply as "agent generation".

For this process to work effectively, the computational model constructor first needs to construct the conceptual model of the worker and then translate that into the computational model by searching through the functional dimension to identify the conceptual identifiers of all components associated with the objectives. It then filters this list down to only those components that are associated with the parts of the environmental contexts with which the agent is associated by searching for those

components in the contextual structure. Once all these components have been identified, the model generation engine identifies all usage mechanisms in each aspect of those components for each knowledge dimension assigned to the worker. This conceptually identifies all the skills, capabilities and processors that the agent model must have to work with those dimensions and components.

Figure 31 illustrates the agent generation mechanism through the example of the air traffic controller from the case study from Chapter 5. The air traffic controllers are placed in a workspace that consists of air traffic control procedures and the radar and voice radio equipment. These components have associated with themselves certain usage mechanisms that are associated with the required set of skills in any worker interacting with these components. For example, the air traffic controller has to possess a skill to follow procedures. Similarly, the usage mechanisms of the radar equipment necessitate skills to read the radar screens. There are certain other analytical skills that the air traffic controller is supposed to have by virtue of the specific procedures that exist in its work environment; for example, certain procedures necessitate the ability to predict conflicts and to calculate distance between points defined by the coordinates of two aircraft. These skills build up on each other and have these relationships specified within their object-oriented models; they also declaratively list the requirements for the skills that they build on to enable the system model constructing facility to assure the presence of those skills on the agent model.

The air traffic controller models are assumed to be interacting with the functional and the contextual dimensions. This necessitates that the air traffic controller models be equipped with skills and capabilities to interact with these dimensions, i.e., draw

inferences from them to understand their work environment and solve their work problems. For example, the air traffic controller is equipped with an ability to read the functional dimension and assess which procedures can be used to achieve his or her current goals. This air traffic controller model, once generated, can be directly used in simulations for operational analysis

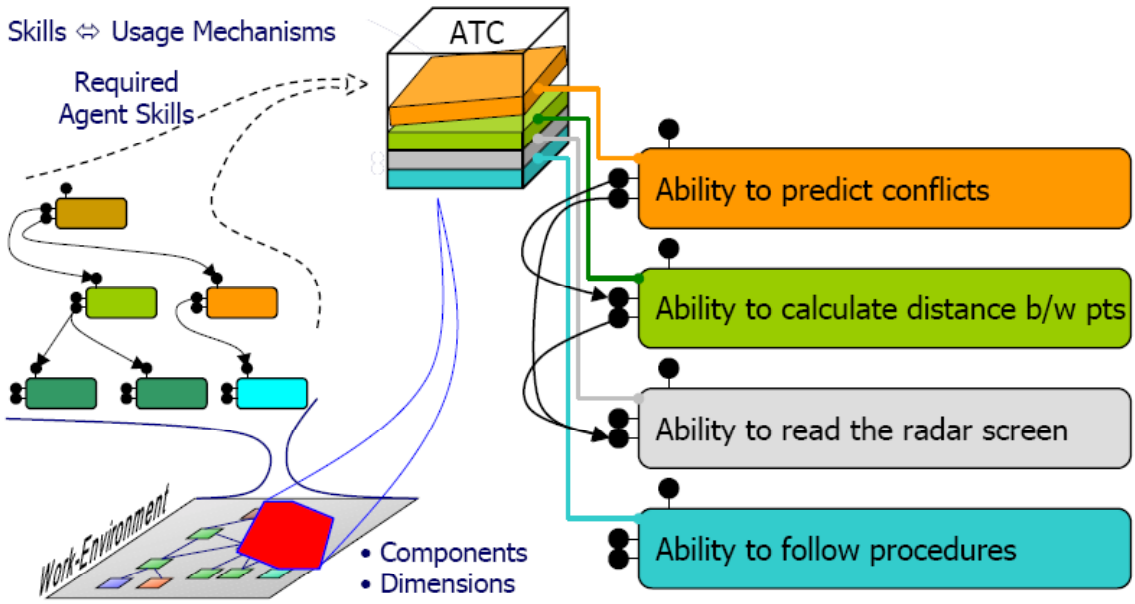

Figure 31: Agent generation mechanism

4.4 Enforcing Environmental Constraints

One of the primary challenges in simulating socio-technical systems is: how does one enforce the constraints of the work environment? For example, how does one enforce that the worker cannot access parts of the work environment that in reality are out of his or her reach? The cognitive engineering point of view would make the complete structure of the work environment visible to the workers so that they can derive the work-tasks best suited for the current situation and the structure of the environment. With the use of such a model in the simulation, the worker would be able to access any part of the environment without any restrictions, and thus do things that he or she could not do in reality. This section discusses how the simulation architecture enforces environmental constraints using the contextual dimension of the model of the work environment.

The challenge (as discussed in challenge 4 in §4.1) is constraining the worker from the point of view of the environment, while allowing the worker models to exercise their creativity and subjective knowledge (which may be inaccurate). This thesis' solution is to use the knowledge specified in the contextual dimension and the system's design specification about workers' assignment to specific contextual nodes (§3.1.2.2) to develop hard constraints on the accessibility of the work environment.

When creating a system model, each worker is associated with one or more contextual nodes in the contextual dimension. The set of these contextual nodes define the context of the worker. During the course of the simulation, the worker's context may change, for example, due to physical movement to some other location in the environment. In such a case the set of associated contextual nodes dynamically changes to reflect the change. These associations are changed by work environment monitoring daemons that are

responsible for maintaining associations of a specific set of contextual nodes. These daemons are continuously running C++ programs that are also time advanced by the simulation engine in the same fashion as the agents and the components in the system. However, these daemons have a global access to the state of the simulated system, unlike the components or the agents, and they can determine when a particular agent's association to a contextual node has to be changed. The logic for such monitoring is encoded into the daemon by the system developer through the use of C++.

The simulation environment enforces that the worker is able to only access properties and usage mechanisms of environmental components that are in the contextual nodes that the worker is associated with, and in the contextual nodes that are downstream in the contextual hierarchy (§3.1.2.2).

4.5 The Simulation Platform

This thesis builds on an existing simulation platform called the Reconfigurable Flight Simulator (RFS) (Ippolito and Pritchett, 2000; Lee, 2002). This simulation platform is meant for agent-based modeling and allows developers and designers to create their own models of agents using the object-oriented design approach in C++. RFS has a fairly general and extensible architecture that can be built on to incorporate novel conceptual constructs suitable for complex-agent based simulations.

RFS was originally built for simulating flight and air space systems, including aircraft, spacecraft, pilots, air traffic controllers and other similar entities encountered in the air and space domain. Figure 32 shows the original architecture of RFS that includes three different kinds of agents (referred to as objects in the figure): vehicle, IO (Input-Output)

and CEM (Controller, Event and Measurement) objects. Vehicle objects, as can be easily guessed by their name, were meant to simulate ground, air and space vehicles; IO objects were meant to simulate human interface panels for these vehicles; and CEM objects were meant for simulation utilities or for modeling complex agents such as human performance models. As is apparent, this modularization was intended to serve the original purpose of flight simulations, where an IO object could only know about specific vehicles, vehicle objects could know about all their IO objects, and CEM objects could access anything within the simulation. The simulation maintained a list for each of these types of objects and used these lists to control access of these types and to advance the simulation by iterating through these lists and advancing the time for each object on these lists.

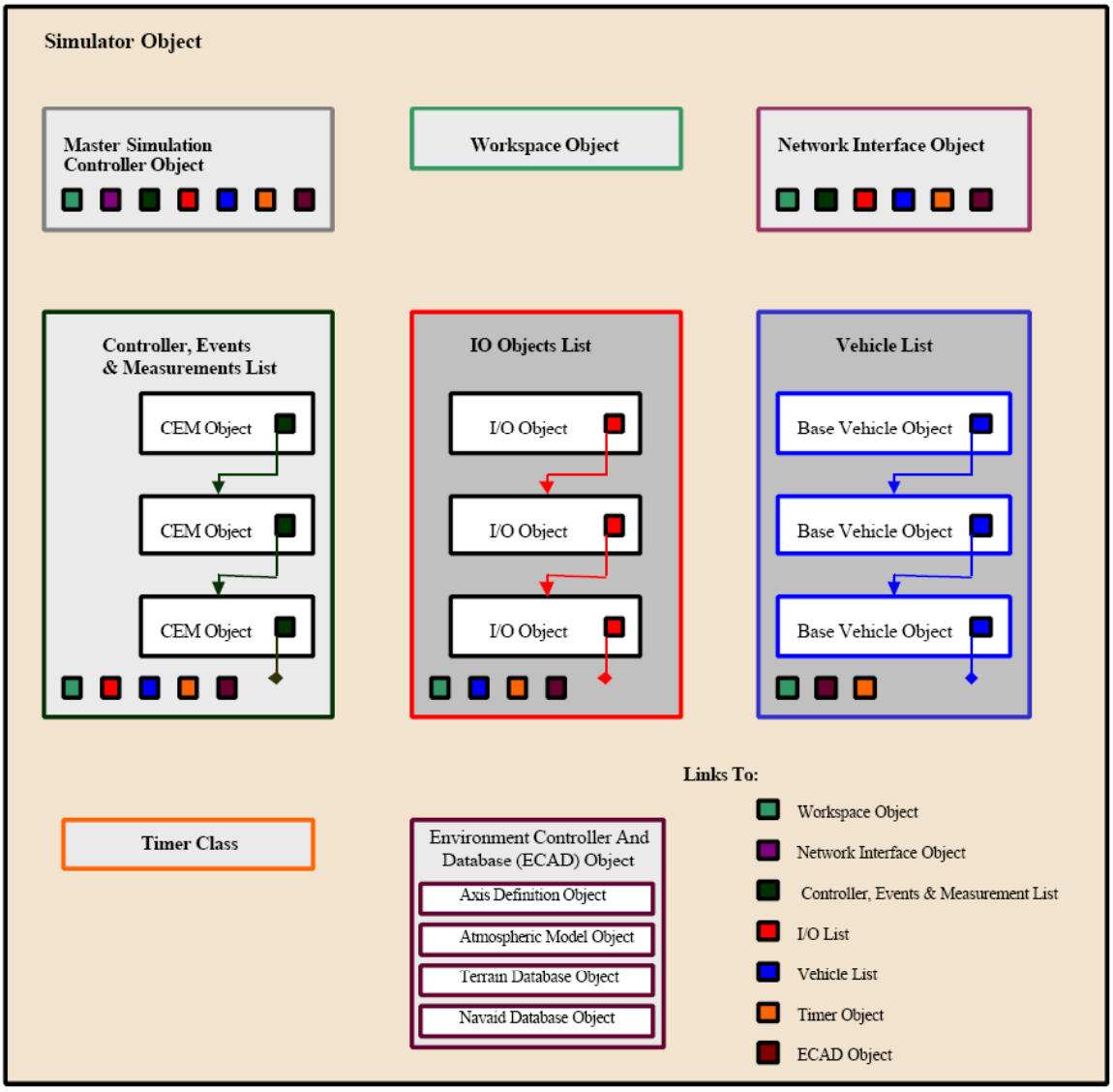

Figure 32: Architecture of the Reconfigurable Flight Simulator (RFS)

Even though the modularization in this architecture draws a very domain-specific picture of the RFS simulation platform, at both the conceptual and object-oriented levels the modules are fairly general and can be extended to domains fundamentally different from aviation.

The simulation platform provides a powerful set of features that allow for high fidelity simulations of disparate entities, both individually and collectively. To enable such simulations, it provides a very flexible and efficient timing and synchronization mechanism that allows for hybrid, i.e., combination of discrete event and continuous time, fast-time simulations. This mechanism allows each individual entity to control both their own time advance as well as of the other entities on which they depend. This provides for autonomous control of dynamics, reduction of computation error, and fast advance of simulation while assuring that important events are not missed. The CEM object in this architecture is modifiable and extensible to represent any kind of object of agent with any kind of internal dynamics or internal architecture. This object was extended to represent agents and environmental components in this thesis. Furthermore, there is a pre-established mechanism for inter-agent and inter-object interaction. This interaction is independent of any particular model, and, at the computational level, it does not require inclusion of C++ code of interacting model components. This mechanism, known as ODME (Object Data and Method Exchange) in RFS, was extended to represent the mapping between the declarative specified usage mechanisms and properties and their computational implementations in object-oriented models. In addition, this simulation already includes the notion of an environment as an entity outside of an agent. This environment (ECAD in Figure 32) includes things such as a terrain database, axis definitions for spatial properties, atmospheric models and navigation databases.

However, this thesis also required several extensions to RFS. First, RFS views the environment as a collection of states and processes that control the changing of these states. From an agents perspective this is the environment that they sense and act upon,

but, it does not have any notion of work-relevance, i.e., specification of individual components of the environment in relevance to the work of the workers or the work-relevant interrelations between them. Furthermore, there was no notion of structuring the environment with such interrelations.

Furthermore, in RFS the notion of an agent is either very particular, i.e., a vehicle, or very general, i.e., CEM object. In principle, any kind of functionality can be modeled within the CEM object through object-oriented programming. A disadvantage of such generality is the extra amount of effort needed to ensure the models' adherence to conceptual frameworks such as the one developed in this thesis. Model developers would have to make sure that the modularity of object-oriented constructs developed by them maintain the modularization and the semantics of the framework. However, it is also very likely for inexperienced model developers to be lured into developing models that are monolithic, semantically different from the framework, and inextensible. Thus, it was needed to specialize this platform to, first, ensure that when developing computational models the conceptual constructs of the framework are adhered to, and, second, to address the practical challenges presented in §4.1.

The following subsections discuss some of the enhancements made to RFS that are general in nature, i.e., apply to any kind of agent-based simulation platform. §4.2, §4.3 and §4.4 have already discussed the computational and architectural elements that address the challenges specific to the modeling and analysis framework developed in this thesis.

4.5.1 Timing and Synchronization Mechanism

This thesis directly employed the timing and synchronization mechanism of RFS, attributed to (Lee, 2002). It is reviewed here because it is key to a number of models that are developed in this thesis. In his work, Lee devised a timing and synchronization mechanism to support fast-time hybrid simulations in which heterogeneous models can both command their next update time and update other simulated objects synchronously when they need to interact in any way. Such a timing mechanism enables the simulation to advance quickly while maintaining time steps such that each object can ensure that computation error does not build up and no events are missed. This timing mechanism is also key to autonomous operation. By commanding its own update time, a model can have complete control over its internal dynamics.

In RFS, the scheduler facility that manages timing and synchronization calls a specific function on each object, which could be an agent or an environmental component, for its requested time of next update. However, in this thesis each may have its own internal dynamics and, in the case of agents, multiple internal processors which may be running asynchronously and in parallel to each other. Thus RFS' timing mechanism had to be enhanced by replicating it within the models of the environmental components and the agents to schedule updates within their internal dynamics in the same manner as the simulation schedules updates of individual objects.

With the availability of such a scheduler on every component and every agent within the large simulation of the system, they can have very complex, self-governed internal dynamics that, in principle, should not limit any kind of agent architectures or modeling of any kind of internal dynamics.

4.5.2 Metrics and Data Collection

Metrics and data collection, whether quantitative or qualitative, provide input to any type of analysis. Metrics for analysis of a design alternative may be collected at both the level of the components and agents, and at the level of the system. These metrics may be collected as a log of timed reading of a specific measure, or based on events, or via online analysis which combines data from various measures in the system. Furthermore, in many analysis of emergent behavior, metrics evolve with the design. Thus, metrics definition and collection can be a very complex task in itself.

This thesis takes a very simple approach to tackle this challenge. A designer can define any kind of 'metrics collector' that inherits from a base object-oriented class called 'MetricsCollector". The base class can be specialized to record specific metrics at specific times. To collect specific measures, the metrics collector can be tied to any of the properties in each contextual node in the contextual dimension that describes the structure of the work environment. This binding is done in the initialization script of the metrics collector by specifying the path of (a syntax similar to that used by XPath) the property whose value is being recorded. For example, the path specification "`//AC1/RadarData::Latitude_deg`" will record the value of the property "`Latitude_deg`" on the contextual node named "`RadarData`" for aircraft named "`AC1`" which specifies the high level contextual node named after itself. Similar to the agent and component models, the metrics collector also updates at self-determined time-steps at which it can sample metrics within a component or agent.

4.5.3 Ensuring Statistically Analyzable Simulations

For most statistical analysis of output data from simulations, stochastic input to the simulation must come from independent and identical random number streams (Law and Kelton, 1999). To enable statistical design of experiments, this thesis built a random number generator capable of providing 2000 independent and identical streams of uniform and normal random variates with seeds spaced 1 million apart using a prime modulus multiplicative linear congruential generator. Simulation runs which required independent and identical variates were configured to have one of the 2000 non-overlapping streams. Each component or agent model in the simulation that requires any kind of random variate gets its next random number from this random number generator.

4.6 Analyzing Transformations

This section presents an approach that uses the conceptual and computational models, to analyze transformations in socio-technical systems. This thesis recognizes two kinds of transformation analysis: operational analysis and network analysis. Operational analysis is concerned with analyzing the emergent performance of the system arising from the individualistic and interactive behavior of the workers situated in and interacting with the work environment, i.e., understanding the operational characteristics of the transformed system. This thesis takes a simulation-based (specifically agent-based simulation) approach to enable such analysis.

Network analysis, on the other hand, analyzes the form and structure of the system by identifying network dependencies between model elements that are specified in the dimension, component and worker models. Such analysis is meant to answer questions

such as which other components and workers will be affected if a particular component is removed from the work environment. For example, if the radar were removed from an air traffic system, this change would be reflected with a change in the workspace of the worker (i.e., change in the contextual dimension). The following sections discuss how these analyses can be performed using this thesis conceptual framework and simulation platform.

4.6.1 Performing an Operational Analysis

As mentioned in the preceding section, operational analysis analyzes the evolution of system state through time. To enable such analysis, the previous sections formulated work-relevant models (of both the work environment and the worker) that can be represented and analyzed computationally. Figure 33 shows how this thesis brings these models together for a simulation-based operational analysis. The multidimensional work environment provides the conceptual base for computational models representing all the dimensions of knowledge available to workers. Transformations at the network level are modeled by changing the relationships that network the components. Component level changes are modeled by either changing the composition and structure of their internal model elements, or by changing the object-oriented implementation of their internal dynamics.

The individual workers in the transformed system are then modeled as agents that are aware of these environmental models. These agent models are given limited views of the work environment corresponding to their context. The agent models are equipped with skills acquired through training, and also with those capabilities that are intrinsic to them.

This creates a complete model of the transformed system, which is then fed to a simulation engine to visualize and analyze the system performance. The simulation engine is also provided the operating scenarios in which the system is to be analyzed. Metrics are collected during the simulation to analyze the emergent performance of the transformed system. The designers and analysts may then draw conclusions about the design and recommend changes to either the work environment or the workers.

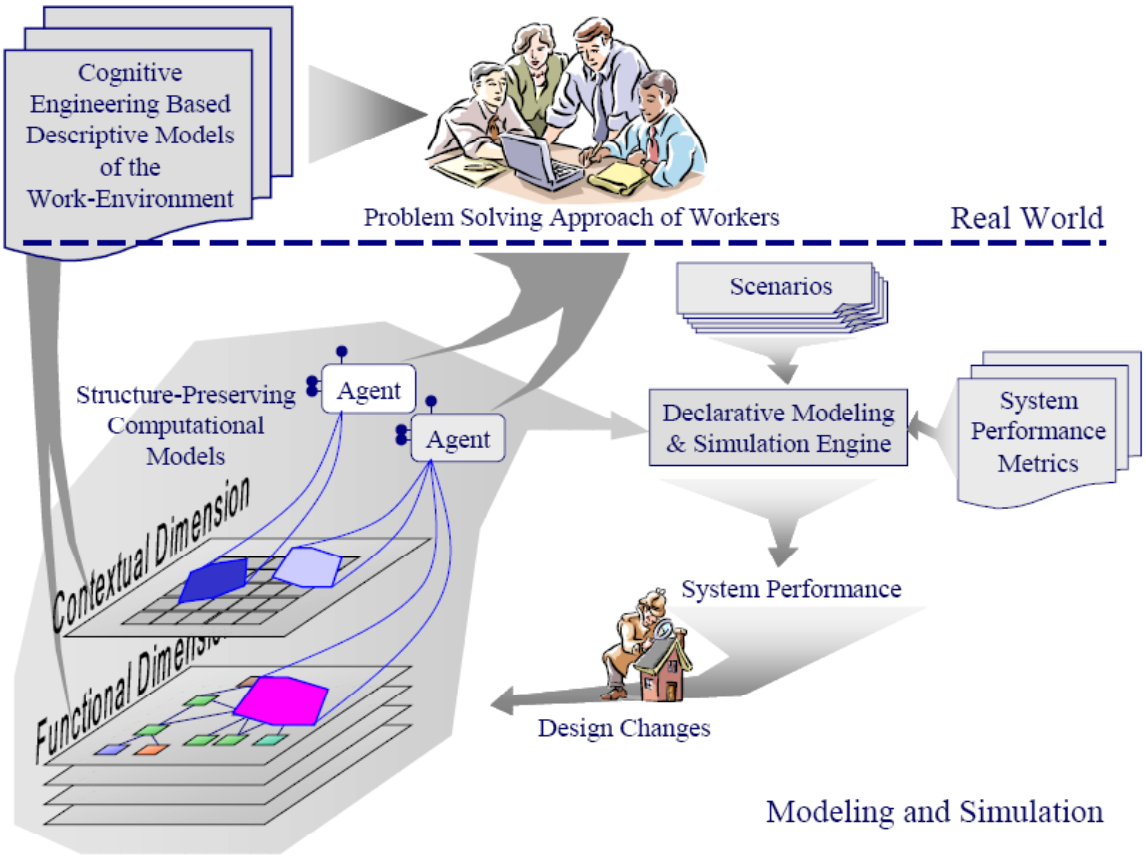

Figure 33: Simulation based analysis of system transformation

This simulation is uniquely suited for this approach to operational analysis of transformations both in the work environment and the worker. That is, it enables analysis of systems that are transformed by changing the skills and capabilities of the worker without changing the work environment, or by making component and network level transformations in the work environment without changing the worker, or a combination of the two.

Note, certain kinds of changes in the work environment may require a concomitant change in the worker. For example, the provision of a new kind of technology in the work environment may require the workers to be trained on it, thus adding to the skills required for the work and thus concomitantly transforming the worker. Thus the generation of agent models from the work environment as described in §4.3 gives this conceptual framework and simulation platform a unique 'work environment centered' operational analysis capability.

4.6.2 Network Analysis

A network level change is made through adding or removing model elements from the system model and/or through changing the structural relationships between them. These model elements can be any of the constructs that were developed in §3.1 and §3.2. That is, they can be individual components and workers, work-relevant relationships between the components, the dimensions, or even the properties, usage mechanism and internal dynamics elements within a component. Since the system model is composed through these model elements, there exist numerous structural and relational dependencies between the instances of these elements that make up the system, and adding or removing

any one of them may break some of them. Thus, while a change will affect the operational performance of the system, there are certain aspects of the system structure that can be analyzed without having to simulate the system. For example, which contextual nodes, i.e., workspaces of workers, would be affected with the removal of a particular component of the system? Or, if a particular component is removed from the workspace of one worker, can it also be removed from the complete system because it may not exist in the workspace of any other worker? There can be numerous such what-if analyses that can be performed simply by querying the declarative specification of the system model as developed in §4.2.1. The structural inference making mechanisms discussed in the following paragraphs can be used very effectively to perform such analysis. Since this thesis has declaratively represented its models using XML, it is very easy to use the logical and structural query language XQuery (Boag, Chamberlin et al., 2005) to perform such analysis based on the semantics of the models developed in the preceding chapter.

Structural and logical queries using XQuery allow for drawing inferences that enable both analyzing the structure of the work environment and using the knowledge of the work environment in the doing of work. For example, the following query can be used on the contextual dimension to list all contextual nodes in the sector ZLA-39 controller's workspace (Figure 24):

```
//ZLA39/*
```

Similarly, the following query can be used to get a list of all properties on the radar in sector ZLA-39 (Figure 24):

```
//ZLA39/RadarData/Properties/*
```

Likewise, if one wants to analyze the impact of removing the radar component "Radar39" from the work environment on the contextual dimension, one can find all contextual nodes in which this component is represented by using the following query on the contextual dimension:

```
//ContextualNode[@WEAComponent="Radar39"]
```

Figure 34 illustrates this thesis' approach to performing a network analysis. First, a system is modeled using the declarative modeling constructs discussed in §4.2.1 and §3.2 for modeling the work-environment and the workers. These declarative models are fed to the declarative model construction engine that, first, constructs the initial structures of all modeled dimensions and, second, updates these dimensions as the simulation advances through time. The analyst can then structurally query these dimensions to examine the dependencies therein. The query engine can be used for such an examination at any point in time to reflect the structure of the system at that particular time. This information can be used by the system analysts to assess the network level impact of any transformations in the system.

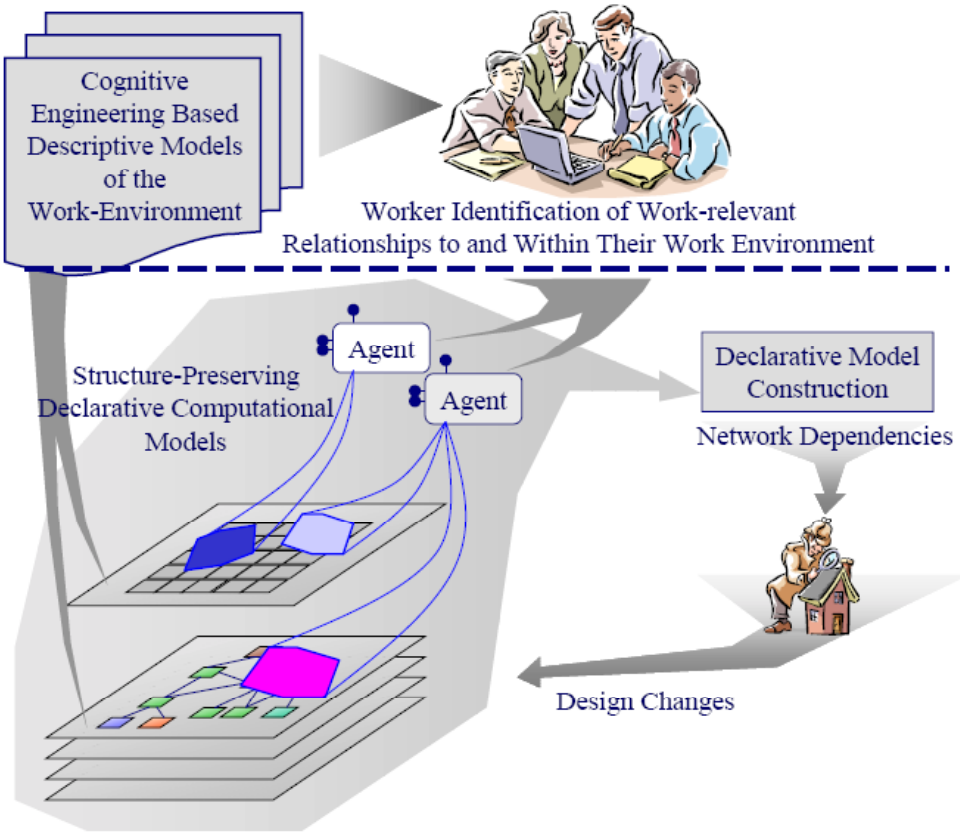

Figure 34: This thesis' approach to performing a network analysis

4.7 Summary and Discussion

This chapter discussed the software architecture and simulation platform that, first, provides for computationally modeling the declarative and behavioral elements of the components and the agents described by the conceptual framework developed in the previous chapter and, second, tackles a number of practical challenges in creating a simulate-able system model from the declarative models.

The primary contributions of the developments described in this chapter were:

1. To facilitate declarative modeling and network analysis, this chapter established an XML representation of the declarative models and developed a mechanism that then automatically assembles, from the individual components' specifications and interrelations, a network-level model of the entire system in XML which can serve to analyze network dependencies between components.
2. To facilitate construction of computational models in a modular fashion that imitates the construction of the declarative system model, this chapter developed two fundamental design-driven constructs objects and facets that are the fundamental units for constructing both the computational model and the conceptual model of the work environment and the agents.
3. This chapter described a mechanism for translating from the declarative model of the system, the components and the agents to their computational model in a manner that represents their declarative construction but provides models of behaviors and internal dynamics that are available in pre-compiled computational models representing individual declarative constructs. This enables the system designers to make network level changes without concerning themselves with reconciling the computational models that encapsulate the behavior of agents or internal dynamics of components.
4. Development of a mechanism for updating the declarative specification of the component, worker and network models to reflect the current state of the system as represented by the simulation. This helps perform network analysis on the system at any given point in time in the simulation.

5. Development of computational modeling constructs that preserve the structure of the system model when incremental changes are made by enabling the designer to only replace computational models of the transformed components. This mechanism allows for making component level changes in internal dynamics and worker level changes in behavior or performance of individual skills, capabilities and processors while not having to concern oneself with the network structure of the system.
6. Developing a mechanism to enforce environmental constraints on agents, even when they may have complete knowledge of the structure of the work environment.
7. Development of a agent-based simulation platform for enabling operational analysis of socio-technical system and the worker, component and network level transformations in them.

CHAPTER 5

DEMONSTRATION: EVALUATING WORK-PROCESSES IN AN AIR-TRAFFIC SYSTEM

This chapter demonstrates the conceptual framework described in Chapter 3 and the software architecture and simulation platform described in Chapter 4 through a case study in modeling, simulating and analyzing an air traffic system. Transformations in this socio-technical system have been enacted through component and network level changes in the work-processes in the work environment of air traffic controllers, component level changes to internal dynamics of aircraft, and worker level changes in their intrinsic capabilities.

5.1 Description of the Case study

The National Airspace (NAS) is continually undergoing transformation. Several traffic management initiatives have been undertaken to increase the capacity of the system while at least maintaining, if not improving, the operational safety of the system. For example, Time-Based-Metering is one initiative that operationally changes the NAS from distance-based control of spacing in air traffic flows to time-based control (Farley, Foster et al., 2001; Mann, Stevenson et al., 2002). This change is enacted through component and network level changes in the work-processes in the work environment of air traffic controllers. This operational change aims to increase the capacity of the airport in terms

of the number of arrivals it can accept in any given time. The success of such a work-processes based transformation is highly dependent on the successful integration of humans, technology, work-processes and information in the system. All aspects considered, this is a case of transforming a large-scale socio-technical system. (FAA, 2005) lists several examples of such operational and structural transformations being made to the national airspace system. This chapter examines such operational changes in NAS, some actual and some hypothetical.

This chapter models air traffic arrivals into the Los Angeles International airport (LAX) with the following procedures: Conflict Avoidance (CA), Miles-In-Trail (MIT), and Time-Based-Metering (TBM). Conflict avoidance procedures are used to separate any two aircraft that have been detected to possibly come too close to each other in the near future. Such an occurrence of the loss of separation is referred to as a 'violation' in this thesis. The MIT procedures are meant to space arriving aircraft by a specific in-trail spacing that is expressed in terms of nautical miles and is provided to the air traffic controllers by the air traffic management units or is agreed upon by controllers of adjoining sectors. These procedures are meant to achieve a specific arrival rate while also spacing aircraft to allow for additional aircraft to merge into the stream. The TBM procedures are also meant to achieve high arrival rates but operate on the measure of time, where desired time of arrival of an aircraft at specific fixes is used as the target to ensure the desired arrival rate.

Figure 35 illustrates the air traffic control system examined here. It shows the eastern part of the airspace for Los Angeles International airport (LAX). This system consists of technological / physical components such as the aircraft (pink dots in figure) that fly

through the airspace to their respective destination airports. In this case study, all aircraft arriving to LAX are termed as arrivals, while all other flights are termed as overflights. The airspace is spatially divided into multiple contiguous sectors (ZLA-39, ZLA-37, ZLA-20, ZLA-19, SCT-FDR), where the boundaries of these sectors are predefined as abstract polygons in the airspace. The air traffic, i.e., all aircraft, in each of these sectors is monitored for conflict-free and procedure-compliant operation by air traffic controllers, each working on their respective sectors. The air traffic controllers are each equipped with a radar screen that displays the traffic in their sector, a voice radio to transmit traffic control commands to the aircraft and receive requests from aircraft, and a specific set of control procedures. These elements constitute the workspace of the controller, with which the controller interacts to achieve his or her goals, i.e., maintain safer operations and ensure compliance with assigned procedures. In this demonstration, this system is transformed through changing the procedures internally and by changing which procedures are in the workspace of each air traffic controller (MIT, TBM or CA).

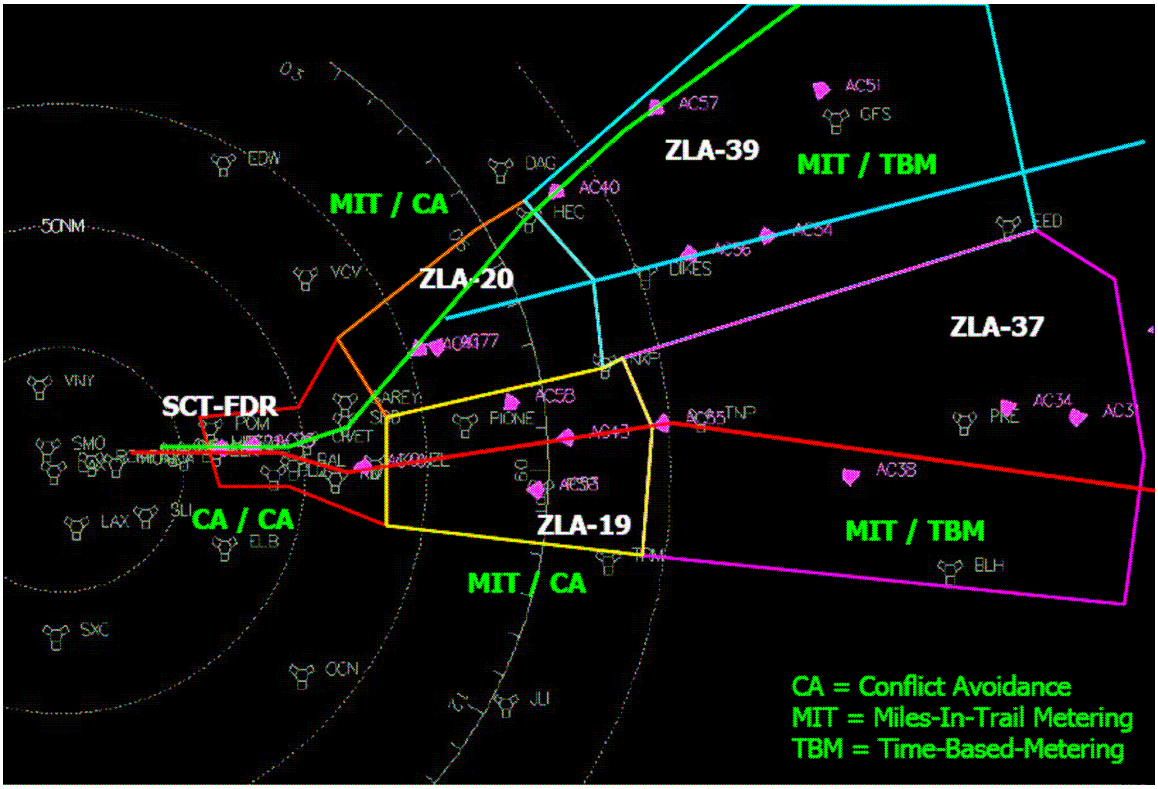

Figure 35: The eastern airspace of the Los Angeles International airport (LAX)

This case has previously been modeled using an agent-based approach, but based on a different conceptual framework. That effort involved four researchers dedicated to developing and testing the models of both the system and its components, designing the experiments, and running the simulations. Additionally, the team was supported by experts in the field of modeling and simulation of human performance and air traffic systems. Another team provided real-world input data and validated the models against observed reality. Unfortunately, this one and a half year effort had limited success in creating validated simulations.

From my analysis of the effort, significant time was spent on developing the models, making them work with each other, and, most significantly, in analyzing the impact of changes to the models of the system's components and the system. The model of each system component and each agent was implemented in monolithic object-oriented classes where the models of the capabilities and skills of the agents and the models of the internal dynamics of environmental components were heavily intertwined. There was limited separation between models of components, work environment and the workers. As a result, one incremental change in the model of any system element could cascade through the system, requiring changes in the models of many interacting components.

The validation efforts in this previous project were the least successful on two accounts:

1. The modeled system manifested two emergent behaviors that are not characteristic of the real world system that was being modeled. These behaviors were observed in a majority of simulation runs.
2. Those simulations that did not manifest those problematic emergent behaviors did not exhibit performance sufficiently close to the real world data for a third party to validate the simulations. The performance was being measured through the use of thirteen different system level metrics, such as the number of separation violations in each controlled sector of the airspace, the average distance traveled in each sector and the average time taken by the aircraft in each sector.

There could be a number of causes of these problems.

1. The input data used to configure the simulations could have errors or biases,
2. The individual models of the system components may not adequately represent aspects of behavior they were intended to represent,

3. The intended aspects of component behavior described by the models may not have included some dynamics critical to emergent system behavior, and
4. The output data against which the system was being validated could be wrong or biased.

Though these factors were identified as possible causes during the previous simulation effort, they could not be tested in practical time due to the limitations of the model forms and their implementation in a simulation platform. Hypothesizing that the use of a structure-preserving and transformation-analysis-oriented modeling and simulation approach, such as that afforded by this thesis, should enable the analysts to analyze these factors in lesser time and more systematically, it was chosen to remodel and re-simulate the system to demonstrate the effectiveness of this thesis' conceptual framework and simulation architecture.

The primary challenges for this effort were to:

1. Eliminate unwanted and unrealistic emergent behaviors from the simulations, and
2. Create a valid simulation, or identify causes of discrepancies between simulation behavior and output data that go beyond the scope of the modeling and simulation capability.

In trying to meet these challenges, this work demonstrates the conceptual framework and simulation architecture by:

1. Developing this thesis' conceptual framework-based models of the air traffic control socio-technical system,

2. Explaining emergent behavior through simulation-based evaluation of the set of probable causes at the component, network and worker levels by transforming the system's models to reflect those changes, and
3. Demonstrating how different transformation alternatives could be compared operationally using simulation-based analysis.

This chapter discusses these three demonstration items in this order. The next section develops a model of the system and its components. In §5.3 an intuitive approach to explaining and eliminating the unwanted emergent behaviors is demonstrated. A comparative analysis of the design alternatives is then conducted in §5.4. §5.5 discusses this thesis' efforts and findings in validating the system models. The chapter ends with a summary of the demonstration and a discussion of the efficacy of this thesis' framework and simulation platform for transformation analysis.

Table 4 summarizes the transformations demonstrated in this chapter. Analyses 1 and 2 attempted to model two forms of the current system; their simulated behaviors were compared to measures of the actual system. Analyses 3 and 4 demonstrated worker and network level transformations; their system performance metrics were compared to the Analysis 2 to assess the relative merit of the transformations. Worker level transformations enhanced the accuracy of workers at certain critical tasks, whereas the network level transformation changed the configuration of the system with respect to which work-processes are available in the context of each sector controller. Component level transformations (Analysis 5) further helped examine unwanted emergent behaviors.

Table 4: Summary of analyses, including purpose and work-processes assigned to each controller

	Work-processes to sector assignment					Purpose	
	FDR	**ZLA-19**	**ZLA-20**	**ZLA-37**	**ZLA-39**		**Transformation**
1	CA	CA	CA	TBM	TBM	Validation	
2	CA	MIT	MIT	MIT	MIT	Validation	
3	CA	MIT	MIT	MIT	MIT		Worker-level
4	CA	MIT + CA	MIT + CA	MIT + CA	MIT + CA		Network-level
5	CA	MIT	MIT	MIT	MIT		Component-level

5.2 Modeling the Air Traffic Management System

This section discusses a representative set of models from the case study to demonstrate how a socio-technical system is modeled using the constructs of this thesis. In previous efforts a number of model specifications were elicited from subject matter experts. This knowledge included specifications for components of the system work environment and for air traffic controller behavior. Components in the work environment included the aircraft, the surveillance and communication equipment used by controllers, the physical structure of the airspace as defined by its division into the sectors, and the air traffic control procedures (work-processes) used by the controllers (pilots were assumed to follow air traffic control clearances exactly, and thus were not modeled in detail). The specification of controller behavior included how they prioritize between tasks and their limitations believed to be intrinsic to human cognition. Apart from the components of the system, the specifications also included knowledge of the structure of the work environment, primarily with respect to the physical structure and the contextual structure.

These specifications were reused for modeling the air traffic management system using this thesis' framework. The following subsections discuss these models; a more detailed description is given in Appendix A.

5.2.1 Air Traffic Controller Work-processes

This section models air traffic control procedures and regulations as work-process components of the work environment. The Federal Aviation Administration and collaborating agencies develop operational procedures to effectively and efficiently manage air traffic, and also develop regulations that the controllers should comply with. In addition, controllers can rely upon informal procedures developed and shared within their immediate community. These procedures and regulations (work-processes) are a part of the system independent of any particular air traffic controller; thus, from an air traffic controller's point of view, these work-processes are a part of their work environment. These work-processes were modeled as work-process components of the work environment.

As discussed in §3.1.3 a work-process has no internal dynamics, two properties (the expression of the situation in which the work-process applies and the process) and one usage mechanism. Represented as a nested tuple (Model (1) in §3.1) the conceptual model of the work-process component looks like:

Work-Process = (Null, (Situation, Process), (Procedure-Following-Mechanism))

Figure 36 shows the listing of one such procedure that describes the situation and the process' properties in natural language. When computationally represented, this component's properties are expressed in XML, including concrete schema specification

for representing the situation and specific work-process instances (excerpts of the work-process shown in Figure 36 are shown in their XML specification in Figure 37 and Figure 38). The usage mechanism 'procedure-following-mechanism' is an algorithm capable of reading these properties, parsing them and using them. For example, this mechanism will make sure that the order of the activities listed in the process is adhered to. Additionally, as discussed in §3.1.3, this usage mechanism will need to access the worker's environmental context to sense the values of the contextual variables that are being used in the situation expression and the process, and to take actions in the context. Thus, the usage mechanism imposes requirements on the air traffic controller model's access to its environmental context.

Heading Merge:

Situation:

1. A conflict is predicted in the future
2. The AC have a common waypoint, i.e. their routes merge ahead.
3. The difference in their heading is more than 15 degrees but less than 135 degrees.

Process:

If conflict time is more than 4 minutes ahead

→

- Spawn Speed control procedure

If conflict time is more than 2 minutes but less than 4 minutes AND

→

- Vector trailing AC by 15 degrees away from the merge
- Speed up the leading AC by upto 50 Kts IAS if it is not already on a speed increase command ELSE slow down the trailing AC by upto 50 knots IAS if it is not already on speed reduction command
- Resume course of vectored AC when appropriate
- Resume speed of the AC whose speed was commanded when appropriate

If conflict time is less than 2 minutes AND
Current vertical separation between them is less than a 1000 ft

→

- Climb higher AC by 2000 ft with VS of 2000 FPM
- Vector trailing AC by 30 degrees away from merge
- Speed up the leading AC by upto 50 Kts IAS if it is not already on a speed increase command ELSE slow down the trailing AC by upto 50 knots IAS if it is not already on speed reduction command
- When vertical separation achieved, command altitude altered AC to approach target altitude with VS of lower AC
- Resume course of vectored AC when appropriate
- Resume flight plan following for AC whose vertical speed was altered
- Resume speed of speed altered AC

If conflict time is less than 2 minutes AND
Current vertical separation between them is greater than a 1000 ft

→

- Spawn Altitude control procedure

Figure 36: Natural language listing of the heading merge procedure for conflict avoidance

```
<KB:Object xsi:type="Proc:typeProcedure" Name="SolveHeadingMerge">

 <Proc:Arguments>
  <Proc:Argument xsi:type="BWT:typeNamedOperand" Type="STRING"
    Name="[.]FirstACInConflict" ArgumentID="FIRSTAC"/>
  <Proc:Argument xsi:type="BWT:typeNamedOperand" Type="STRING"
    Name="[.]SecondACInConflict" ArgumentID="SECONDAC"/>
 </Proc:Arguments>

 <Proc:WorkCondition CheckConditionAt="START" EvaluateToValue="true"
   EvaluateToType="BOOLEAN">
  <BWT:Expression xsi:type="BWT:typeOperator" Name="&amp;"
   Type="BOOLEAN" DLL="" Value="">
   <!-- The given AC must be in conflict at time ahead -->
   <BWT:Argument xsi:type="BWT:typeOperator" Name="areACInConflict"
     Type="BOOLEAN" DLL="" Value="">
    <BWT:Argument xsi:type="BWT:typeNamedOperand" Name="[@PA]FIRSTAC"
     Type="STRING"/>
    <BWT:Argument xsi:type="BWT:typeNamedOperand" Name="[@PA]SECONDAC"
     Type="STRING"/>
   </BWT:Argument>

   <!-- Current heading difference must be greater than 15 degrees -->
   <BWT:Argument xsi:type="BWT:typeOperator" Name="&gt;"
     Type="BOOLEAN" DLL="" Value="">
    <BWT:Argument xsi:type="BWT:typeOperator" Name="fabs"
     Type="DOUBLE" DLL="" Value="">
     <BWT:Argument xsi:type="BWT:typeOperator" Name="-" Type="DOUBLE"
       DLL="" Value="">
      <BWT:Argument xsi:type="BWT:typeNamedOperand"
        Name="[//]RadarData{[@PA]FIRSTAC|STRING}::PtrHeading_deg"
        Type="DOUBLE"/>
      <BWT:Argument xsi:type="BWT:typeNamedOperand"
        Name="[//]RadarData{[@PA]SECONDAC|STRING}::PtrHeading_deg"
        Type="DOUBLE"/>
     </BWT:Argument>
    </BWT:Argument>
    <BWT:Argument xsi:type="BWT:typeValueOperand" Type="DOUBLE"
      Value="15.0"/>
   </BWT:Argument>

   <!-- Current heading difference must be less than 135 degrees -->
   <BWT:Argument xsi:type="BWT:typeOperator" Name="&lt;"
     Type="BOOLEAN" DLL="" Value="">
    … …
   </BWT:Argument>
   … …
  </BWT:Expression>
 </Proc:WorkCondition>
… …
```

Figure 37: Partial XML specification of the heading merge procedure shown in Figure 36, listing the partial specification of the applicable situation.

```
<KB:Object xsi:type="Proc:typeProcedure" Name="SolveHeadingMerge">
</Proc:WorkCondition>
 … …
</Proc:WorkCondition>

<Proc:Process>
 <!-- Define variables -->
  … …

 <!-- Start the process with forking amongst the four conditional
      resolutions -->
 <Proc:Fork NodeID="ResolveHeadingMerge">

   <!-- CONFLICTTIMEAHEAD > 4 mins, spawn speed control  -->
   <Proc:Choice>
    <!-- specify condition expression  -->
    <Proc:Expression EvaluateToValue="true" EvaluateToType="BOOLEAN">
     <BWT:Expression xsi:type="BWT:typeOperator" Name="&gt;"
       Type="BOOLEAN" DLL="" Value="">
      <BWT:Argument xsi:type="BWT:typeNamedOperand"
        Name="[@VA]CONFLICTTIMEAHEAD" Type="DOUBLE"/>
      <BWT:Argument xsi:type="BWT:typeValueOperand" Type="DOUBLE"
        Value="240"/>
     </BWT:Expression>
    </Proc:Expression>

    <!—Spawn speed control procedure  -->
    <Proc:Spawn ProcedureName="SpeedControl" NodeID="spawnSpdCntrl"
      Parallel="true">
     <Proc:Arguments>
      <Proc:Argument xsi:type="BWT:typeNamedOperand"
        Name="[@PA]FIRSTAC" Type="STRING" ArgumentID="FIRSTAC"/>
      <Proc:Argument xsi:type="BWT:typeNamedOperand"
        Name="[@PA]SECONDAC" Type="STRING" ArgumentID="SECONDAC"/>
     </Proc:Arguments>
    </Proc:Spawn>
   </Proc:Choice>
   … …
 </Proc:Fork>
 … …
</Proc:Process>
</KB:Object>
```

Figure 38: Partial XML specification of the heading merge procedure shown in Figure 36, listing the partial specification of the process.

This case study used three work-processes:

1. Miles-in-trail (MIT): These work-processes help the air traffic controllers space aircraft on the same route by a given distance. In real-life the desired distance is provided to the controllers by outside traffic management units.
2. Time-Based-Metering (TBM): In time based metering the Traffic Management Advisor (TMA), a technological aid, provides the controllers with 'delay times' that specify when each aircraft should arrive over a specific fix. The time-based-metering work-processes are meant to help the controller slow and space the aircraft to 'absorb' their delay times.
3. Conflict Avoidance: These work-processes are meant to help the air traffic controller avoid conflicts between aircraft. A conflict is a situation when two aircraft come closer than five nautical miles horizontally and one thousand feet vertically above an altitude of 18,000 feet, and three nautical miles horizontally and one thousand feet vertically below 18,000 feet.

The assignment of work-processes to each controller was one way of specifying the system's network level structure. For example, when exercising MIT control, the controllers of the higher sectors, i.e., ZLA-37, ZLA-39, ZLA-20 and ZLA-19, were given the MIT work-processes, while the SCT-FDR controller was given the Conflict Avoidance work-processes. On the other hand, when exercising TBM control, controllers of sectors ZLA-37 and ZLA-39 were given TBM work-processes while the other three were given conflict avoidance work-processes.

5.2.2 Multidimensional Model of the Work environment

The system modeled in this demonstration consisted of five air traffic controllers, each controlling the airspace in one of the five contiguous sectors on the eastern approach to the Los Angeles International Airport (LAX) (Figure 39). There are a number of flights that fly through these sectors: some of them are arrivals into LAX (as shown by the red and green lines that extend from the right hand side of Figure 39 to the left hand side); others are considered 'over-flights'. Each sector controller has a display of flights in his or her sector, and a voice radio by which he or she issues commands to pilots onboard the aircraft. The controllers have little knowledge of the aircraft in other sectors and do not frequently communicate with other controllers; therefore, the other controllers and aircraft outside their sector are not considered to be in their context.

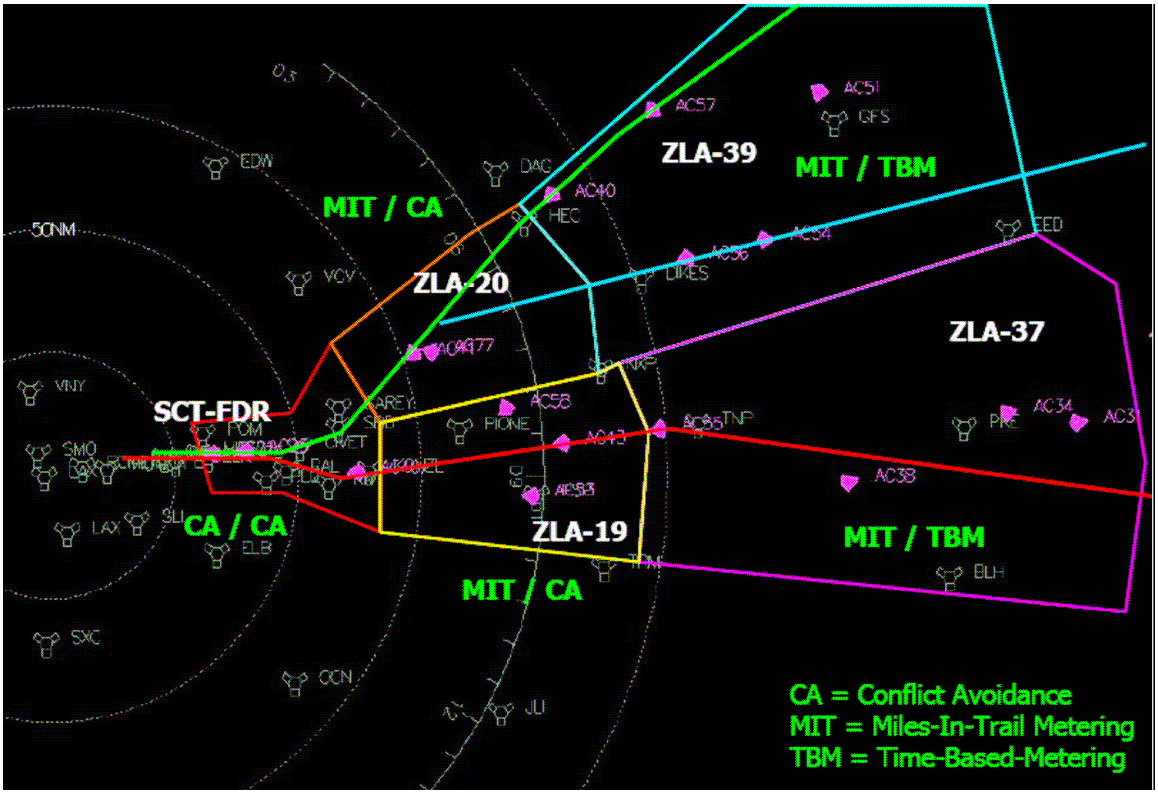

Figure 39: Sectors and arrivals at LAX

Based on this specification we know that the components in the work environment of the air traffic controllers include the aircraft, the surveillance and voice radio equipment and the work-processes' and can be represented in the form given by model (2) in §3.1.1. The reader should note that not all these components have the same lifespan as the controller: aircraft appear in the work environment according to their schedule and then exit the airspace. Thus, this is a dynamic work environment.

This section models the work environment of the air traffic controllers on the two fundamental knowledge dimensions described in Chapter 3: the contextual dimension and the functional dimension. For the purpose of this study, no other dimensions were

deemed necessary because all the controllers need to know is: what is the current situation (which components of the work environment are in their context and what are the states of their properties), and which work-processes they can use to achieve their goals.

5.2.2.1 The Contextual Dimension

Viewed from top-down, the contextual dimension is constructed by assembling contextual nodes for each component of the work-environment and then associating them through contextual-compositions. In this case, the contextual dimension includes one contextual node for each control sector, which further subsumes the contextual nodes for the surveillance equipment, the control equipment, the wind measurement equipment, the work-process information system and the flight strips management system (Figure 40). These further subsume contextual nodes containing those parts of each component available to the contextual node.

Viewed from the bottom-up, the model of each component of the work environment includes a set of properties and usage mechanisms. These attributes are grouped into aspects that map them onto the contextual dimension of the work environment (§3.1.2.2). For example, let us consider the aircraft component. For the purpose of air traffic control, the work-relevant properties of the aircraft may be summarized as:

1. Latitude,
2. Longitude,
3. Altitude,
4. Ground speed,
5. Vertical speed,
6. Heading,
7. Flight Path Angle, and
8. Flight Plan

The aircraft has other properties such as:

1. Number of passengers on board,
2. Bank angle, etc.

that are not useful for the work of air traffic controller and hence for the purpose of this analysis, and therefore they are not included in the contextual dimension.

Each aircraft model has a set of contextual aspects that map these attributes onto the contextual nodes comprising the contextual dimension. The surveillance equipment includes a set of contextual aspects of each aircraft within its monitored airspace through a contextual-composition relationship (Model (10) in §3.1.2.2). The internal dynamics of the surveillance equipment represents a daemon (ref §4.4) that makes sure that at any given point of time in the simulation the contextual nodes of all aircraft in the sector are included in the contextual node of the surveillance object. Similarly, contextual nodes are created for flight strips available in the context of a controller (Figure 40).

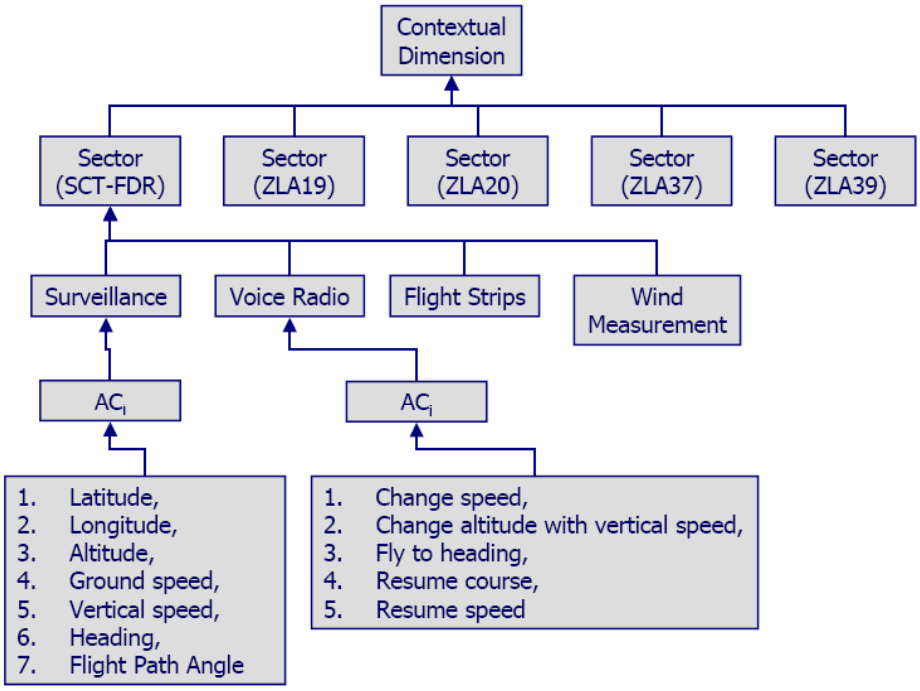

Figure 40: Part of the contextual dimension in the case study

The usage mechanisms of an aircraft include:

1. Change speed,
2. Change altitude with vertical speed,
3. Fly to heading,
4. Resume course, and
5. Resume speed.

In the real world, these usage mechanisms are available to the pilot through controls in the cockpit or through flight management systems. For example, pilots tune their communication equipment to the frequency assigned to the control sector that they are flying through and thus come into the contextual dimension of the controller, where they can be given commands by the controller. In this case study, these pilot-controller communication mechanisms were assumed not to be important for the analysis and these usage mechanisms are directly made available in the context of a controller by mapping them over to their contextual dimension through their voice radio equipment.

5.2.2.2 The Functional Dimension

As described in §3.1.2.1, the functional dimension relates the environmental components to the goals through means-ends-constraints relationships; in other words, it identifies them as affordances and constraints towards one or more goals. The construction of the functional dimensions starts with identification of the goals (Figure 41).

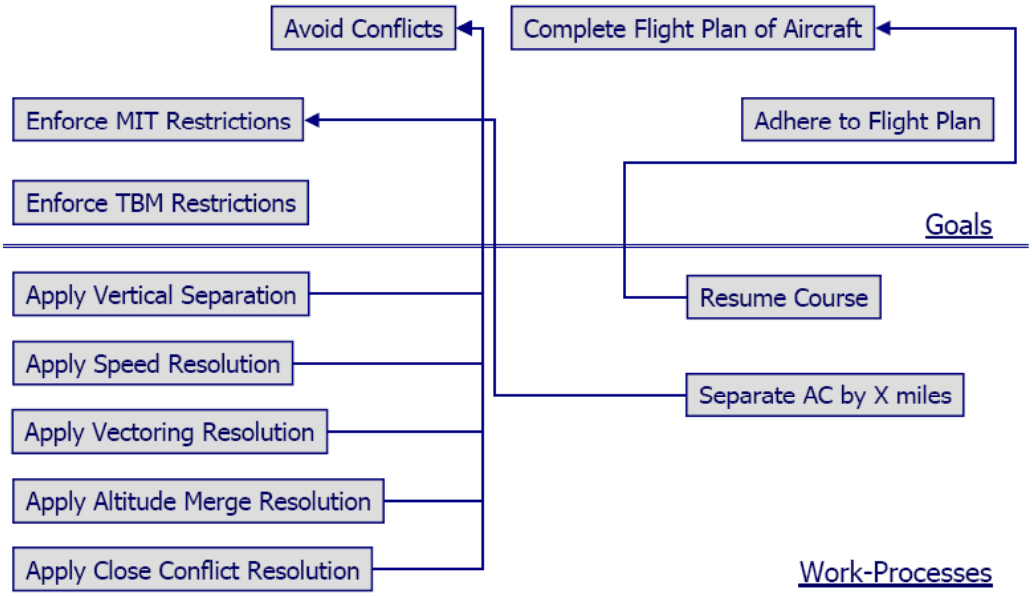

Figure 41: Relating goals and work-processes via the functional dimension

Once the goals have been identified, the environmental components are tagged as means and constraints. In this case we have two kinds of environmental components: technological artifacts and work-processes. For the sake of brevity of this discussion,

Figure 41 shows a partial model including only the goals and the work-processes; the artifacts are included in the full model in an identical manner as the work-processes. In the figure, the associations with arrowheads identify that the component at the tail of the arrow is a means to the work-objective at the head of the arrow. If the arrowhead is a dot, it identifies the component at the tail of the arrow as a constraint.

5.2.3 Environment-Centered Construction of Agents

In this section, models of the air traffic controllers are automatically generated from the work environment model and the assignment of goals and context to workers. Air traffic controllers are the only workers modeled here. There are five of these workers, one for each of the control sectors. They are each allotted to a specific sector in the contextual dimension (Figure 40), thus making the appropriate contextual node (and any it subsumes) available to each controller. They are each allotted a subset of goals from the set shown in Figure 41. For instance, when the system design represents an MIT configuration, the air traffic controller for sector ZLA19 (one of the four higher sectors that enforce MIT restrictions between arrivals) is allotted the following objectives: Enforce MIT Restrictions, and Complete Flight Plan of Aircraft. For an environment-centered design, making these allotments is the only activity required when designing the system.

However, as noted in §2.2, §3.2 and §4.2.2.3, an agent also needs to be constructed as a configuration of skills, capabilities and processors. This thesis' simulation platform is capable of building the agent based on the system-level design. Specifically, using the model composition architecture discussed in §4.3, the model construction engine first

identifies all environmental components associated with the allotted goals on the functional dimension and with all components in the contextual dimension that are associated with the allotted context of the worker. The usage mechanisms of these components are identified and the skills and capabilities associated with them are picked out from the computational model registry and added onto either an empty agent model or to a template agent model which may be hand-picked by the designer. When a template model is picked by the designer, the model construction engine identifies those skill and capability implementations from the computational model registry that correspond to the template. These skills and capabilities are then aggregated into the agent. At this point the agent model is ready to be used in the simulation for operational analysis; in addition, the assembled list of skills and capabilities can be examined during network analysis to examine the feasibility of the set for the intended worker and to identify training and information requirements for the worker.

Table 5 shows the full set of skills and capabilities in the controller for sector ZLA-39 when given MIT procedures. Each of these skills and capabilities was coded using C++ into facets that can be aggregated into the agent model (§4.2.2.1).

Table 5: The full set of skills and capabilities of the controller of sector ZLA-39 when given MIT procedures

	Skill Name	Skill Description
1	AreACInConflict	Check if two aircraft are in conflict
2	GetIfCommonWaypoint	Check if two aircraft have a common waypoint
3	GetDistanceBtPts	Get horizontal distance between two points
4	GetHdgBtPts	Get heading between two waypoints, as measured from the North
5	GetIfFailedResolution	Makes sure if a particular resolution failed for given aircraft in a given conflict
6	IsPointInSector	Check if a given point is in sector
7	GetDstncToMergePoint	Calculate the distance between an aircraft's current position and the merge point of that aircraft and another aircraft.
8	GetHdgFromMergePoint	Get the orientation of an aircraft from the merge point of that aircraft and another aircraft
9	ChangeSpeed	Command a given aircraft to change speed
10	ChangeAltWithVS	Command a given aircraft to change altitude with a given vertical speed
11	ChangeHeading	Command a given aircraft to change heading to a given heading
12	ResumeCourse	Command an aircraft to resume course
13	ResumeSpeed	Command an aircraft to resume waypoint speed
14	ResumeAltitude	Command an aircraft to meet waypoint altitude restriction
15	Wait	Wait for a given amount of time
16	GetDoubleProperty	Read a variable from the context
17	GetFlightPlan	Read flight plan of a given aircraft from the context
18	MonitorConformance	Monitor boundary conformance for all aircraft sent off-course by the controller
19	MonitorTraffic	Monitor traffic in sector for possible conflicts
20	IsChangeConflictFree	Judge if a particular maneuver for a particular aircraft would be conflict free in given time frame
21	CalculateLatLongAltAtTimeAhead	Calculate the position of an aircraft in given future time
22	IsACReadyToResumeAlt	Make sure that the aircraft can resume altitude without creating future conflicts

Table 5 (continued)

23	IsACReadyToResumeSpeed	Make sure that the aircraft can resume speed without creating future conflicts
24	IsACReadyToResumeHeading	Make sure that the aircraft can resume course in two dimensions (not vertically) without creating future conflicts
25	IsACReadyToResumeCourse	Make sure that the aircraft can resume three-dimensional course without creating future conflicts
26	IsACOnAltChange	Check if the aircraft has been commanded altitude changes
27	IsACOnHeadingChange	Check if the aircraft has been sent off-course
28	IsACOnSpeedChange	Check if the aircraft has been commanded to change speed
29	MonitorMITTraffic	Monitor arrivals for in-trail spacing violations
30	IsACInMITViolation	Check if a particular aircraft is closer to any other arriving aircraft than the required in-trail spacing
31	GetViolationDistance	Calculate the distance that a particular aircraft will have to absorb to avoid in-trail separation violation
32	GetDistanceToDestination	Calculate aircrafts along track distance from destination
33	GetHeadingFromDestination	Calculate orientation of a given aircraft from the destination
34	IsPastILSMerge	Calculate if an arrival aircraft is past the ILS merge point for the destination
35	GetHeadingFromILSMerge	Calculate the orientation of the aircraft from the ILS merge
36	GetDistanceToILSMerge	Calculate the along track distance of an aircraft from the ILS merge point
37	GetConflictTimeAhead	Calculate time to conflict
38	FollowProcedures	Follow the procedures in the context

The agent models were constructed from the template for the rudimentary human performance model provided in §3.2.4. This template was populated with the skills and capabilities that are specifically needed of an air traffic controller based on the specifications just discussed. Except when a worker-level transformation was examined (described in §5.4.2), the activity parameters were as shown in Table 6.

Table 6: Specification of activity parameters for the air traffic controller model

Agent Type	Limited resources (Max Resources = 7)					
Source of Variability	Duration		Accuracy	# Resources		Resource Acquisition Priority Rating
Distribution	Normal		Uniform	Normal		
Activity	Mean	Stdev	Probability	Mean	Stdev	
Monitor Traffic for Conflicts	6	4	0.8	3	1	5
Monitor Traffic for MIT Spacing	3	2	0.8	3	1	4
Monitor Traffic for TBM Compliance	3	2	N/A	2	1	4
Change Speed	6	2	N/A	4	1	8
Change Heading	6	2	N/A	4	1	8
Change Altitude	6	2	N/A	4	1	8
Resume Course	3	1	N/A	4	1	8
Resume Speed	3	1	N/A	4	1	8
Monitor Sector Boundary Conformance	2	6	N/A	2	1	10
Wait	Var	5	N/A	1	1	1
Follow Procedures	Unlimited		N/A	2	2	1

5.3 Explaining Emergent Behaviors

This section demonstrates how the capability to transform a system model at the component level was used explain the two unrealistic emergent behaviors observed in the previous efforts to model the air traffic control system:

1. Occurrences of unplanned steep descents in flight paths, and
2. Occurrences of unplanned horizontal "loops" in flight paths.

These emergent behaviors could arise from either the aircraft or the controller behaviors, mismatched or untimely interaction between the two, the work-process specifications, or a combination of these factors. Since the previous modeling architecture was not well suited to quick modifications of its models of environment components and agent behaviors, exploring the full set of possible changes to the models would have required significant software modification and a prohibitive duration of development time.

Once the system was modeled using this thesis' approach and implemented over the software and simulation architecture, the level of effort to test several design variables was significantly reduced and a sufficient range of conditions was examined to not only explain the causes of those emergent behaviors but also eliminate them in the simulation, as described in the following sub-sections.

5.3.1 Explaining Steep Descents

The altitude profiles of arrivals using the previous simulation are shown in Figure 42. It was observed in the previous simulation that to avoid conflicts some aircraft would be commanded to either hold altitude or climb to higher altitudes, and then later be commanded to resume course and meet waypoint altitude restrictions. In such situations,

the aircraft would sometimes exhibit steep descents. Approximately 98% of the 11,200 simulation runs, each with approximately two hour of simulated time and an average of approximately 28 arrivals per simulation, manifested this behavior for one or more aircraft irrespective of which procedures (MIT or TBM) were in place.

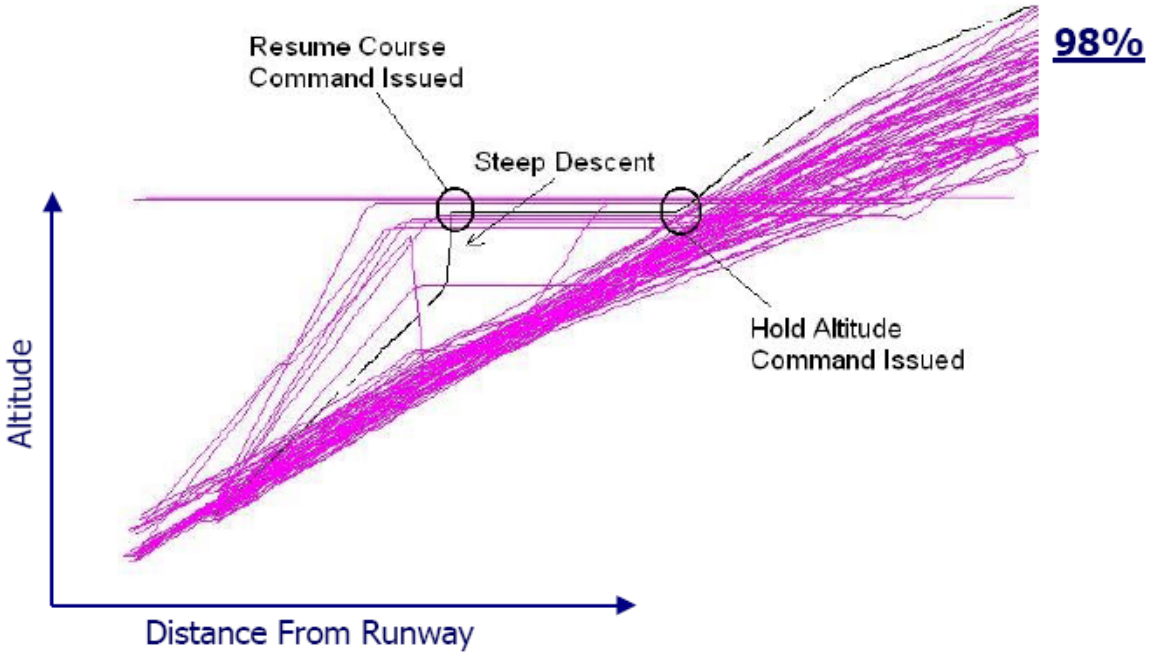

Figure 42: Vertical profile of arrivals using the previous models

As a first step in finding the source of and eliminating these unwanted emergent behaviors, the internal dynamics of aircraft were modified to have internal limits on their descent rates. Changing the system model for such a component level transformation in the system was as simple as replacing the facet that models the basic flight dynamics on

the aircraft model with a new facet that enforces limits on behavior. Although this change corrected the problem of steep descents, it resulted in the aircraft not being able to meet their altitude restrictions (Figure 43), another unrealistic emergent behavior. In addition, the traffic flow within the airspace changed in a direction that led to too many conflicts and too many altitude hold commands. This component level transformation was therefore discarded as a possible candidate for eliminating the unwanted emergent behavior.

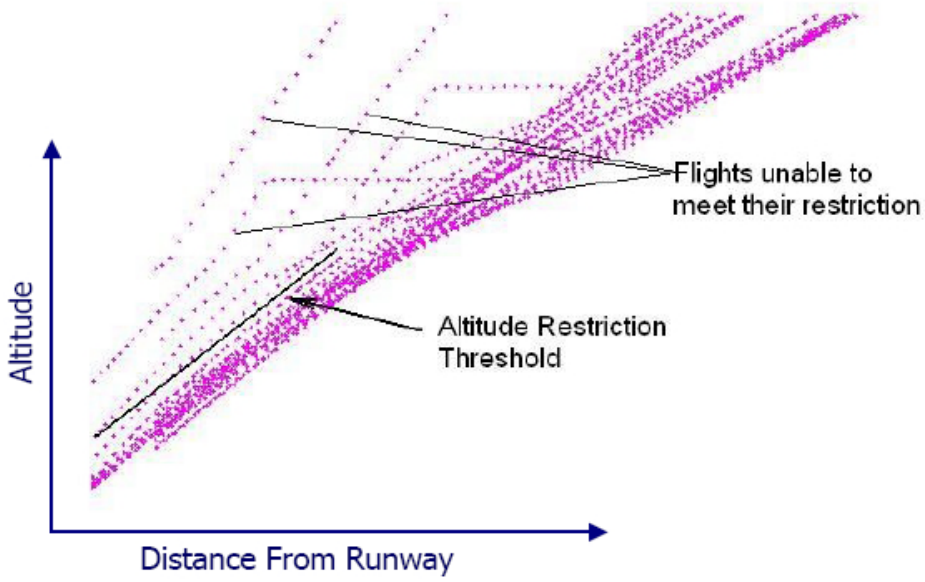

Figure 43: Vertical profile for arrivals when limits on vertical speeds were encoded in aircraft internal dynamics

Since transforming aircraft behavior was not the correct solution to the problem, the next step examined transformations in work-processes. As shown in Figure 42, the aircraft were commanded to hold altitude in order to avoid conflicts. As a result, the aircraft drew close to their next horizontal waypoint while staying at a much higher altitude.

Thus, when commanded to resume course and meet the altitude restriction of their next waypoint, they tended to exhibit very steep descents. This observation was used to transform the work-processes to not hold aircraft at the same altitude for very long, but instead descend aircraft at fixed vertical speeds as soon as they are vertically separated (Figure 44). The overall system performance in terms of separation violations and number of occurrences of steep descents improved with such a change: although the aircraft continued to sometimes exhibit descents at a high rate, these descents were not as drastic and as frequent as before. Examining the recorded behaviors found the remaining steep descents occurred when the aircraft were commanded to descend at the given vertical speed at lower altitudes where the true air speeds are lower. Modifying the work-processes further to schedule the commanded vertical speed by altitude resulted in reduction of steep descents from occurring in 98% of simulations to 0.3% (Figure 44), while also improving system performance in terms of reducing the number of violations recorded per run and the number of total diversions from planned flight paths of aircraft. Figure 45 shows the output of one simulation where one flight was held at a given altitude but, when later brought back on course, its rate of descent was within tolerance.

The effort involved in making these component level transformations in the work-processes was as little as changing its XML representation of which components to include and modifying isolated models of activities, without any cascading interactions between the various component and agent models.

Process to vertically separate two arrivals

(a) Hold altitude of higher aircraft

(b) Recheck every 30 seconds if resuming course is conflict free

(c) Resume course when appropriate

Transformed process to vertically separate two arrivals

(a) Hold altitude of higher aircraft until vertical separation is achieved

(b) Once separated vertically, descend higher aircraft to next waypoint altitude at vertical speeds less than that of lower aircraft. When assigning vertical speed, compensate for the following

(1) If true airspeed of aircraft is greater than 250 knots, assign 2000 feet per minute

(2) If true airspeed of aircraft is less than 250 knots, assign 1500 feet per minute

(c) Recheck every 30 seconds if resuming course is conflict free

(d) Resume course when appropriate

Figure 44: Vertical separation procedures before and after transformation

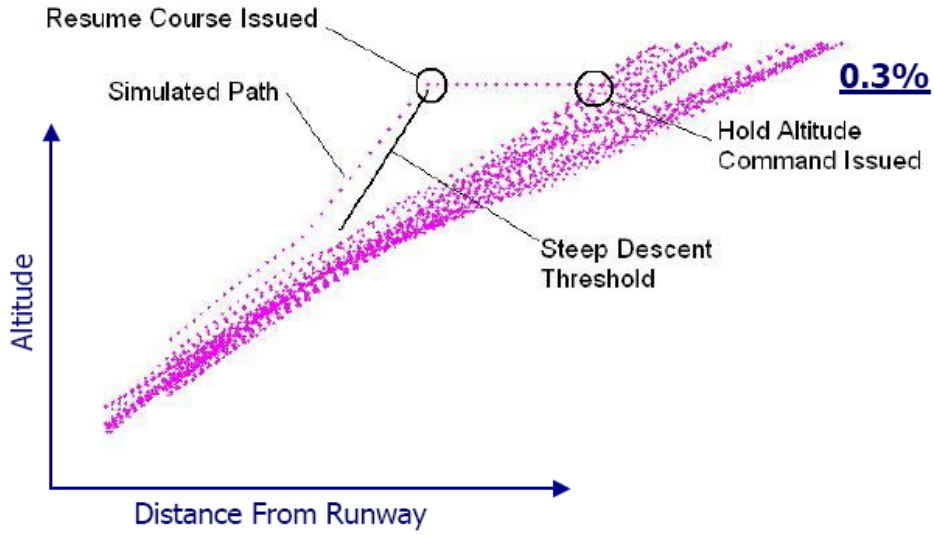

Figure 45: Vertical profile of arrivals when air traffic controller work-processes were changed to not command high vertical speeds

5.3.2 Explaining Loops

It was observed in the previous modeling effort that, to avoid conflicts, some aircraft were vectored off their routes and then be commanded to resume course. In such situations the aircraft would sometimes exhibit a "horizontal looping" behavior (Figure 46) where the aircraft turns back to the waypoint it was originally enroute to. Approximately 51% of the 11,200 simulation runs manifested this behavior.

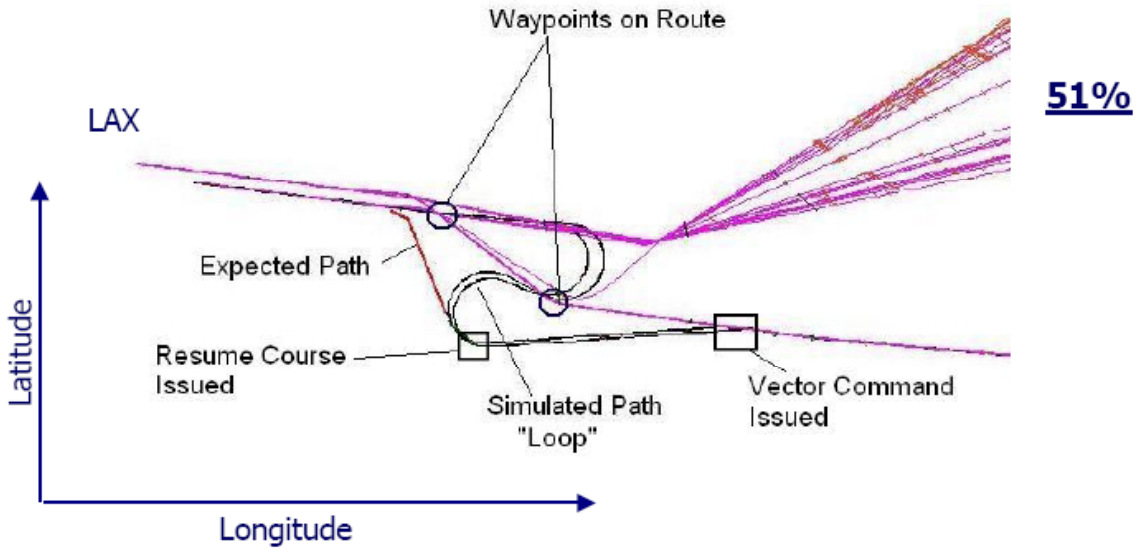

Figure 46: Horizontal profile of arrivals in previous simulation exhibiting the horizontal looping behavior

Similar to the case of steep descents, any of the aircraft and controller behavior and the work-processes models could account for such a behavior. Each one of these possible

causes were examined. Examination of the event logs showed that the aircraft, when commanded to resume course after the earlier deviation from it, resumed their path to the waypoint that they were enroute to before they were vectored off. If that waypoint was now behind them they had to horizontally loop back to reach it. This was obviously a mismatch between the internal dynamics in the flight management system (FMS) of the aircraft and the expectations implicit in the controller's command to resume course. Figure 47 illustrates the internal mechanism of the aircraft's model of its FMS. The pink tag on the usage mechanism of the aircraft that is used to command the aircraft to vector off route at a given heading invokes the heading override behavior in the internal dynamics of the aircraft. This event-based (responsive) behavior further changes the aircraft's autonomous waypoint following behavior to steer it off the route. When a resume course command is issued through the use of the corresponding usage mechanism (identified by the green tag), it invokes a course-resumption behavior in the internal dynamics that again affects the autonomous waypoint following behavior of the aircraft. It was noted that the previous model's course-resumption behavior always reverted to the waypoint before the off-route deflection. On the other hand, the controller assumed that the aircraft would resume course towards the next waypoint in its flight plan that has a bearing between +90 degrees and –90 degrees relative to its current heading, without the controller needing to explicitly direct the aircraft to this new waypoint.

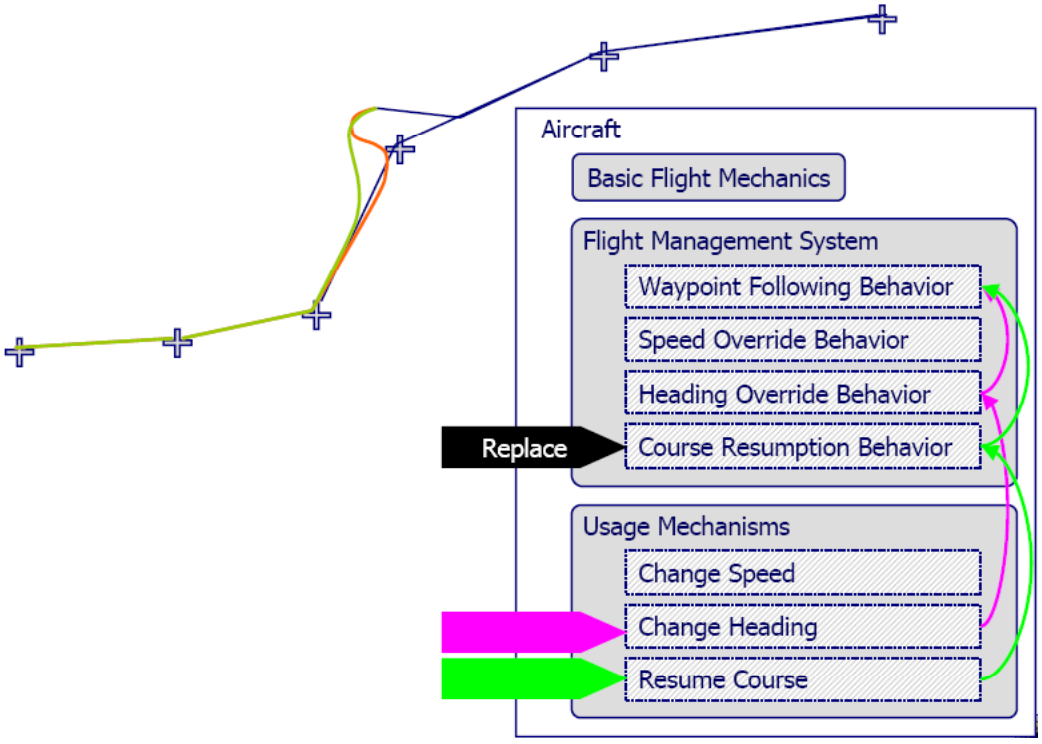

Figure 47: Illustration of the internal mechanism of the aircraft model that leads to horizontal looping behavior

In order to test this hypothesis, a component level transformation was introduced in the system through changing the course resumption behavior in the internal dynamics of the aircraft model to match the assumptions of the controller. Using the architecture of this thesis, this transformation was easily introduced by simply replacing the facet that implemented the course resumption behavior, within the internal dynamics of the aircraft, with a new one that exhibited the desired FMS characteristics. Due to the modular nature of the model elements, this change did not require any concomitant changes in any other elements of the aircraft or the system models. Furthermore, this component level change

completely eliminated the "horizontal loop" (Figure 48), thus also explaining the cause of this unwanted emergent behavior.

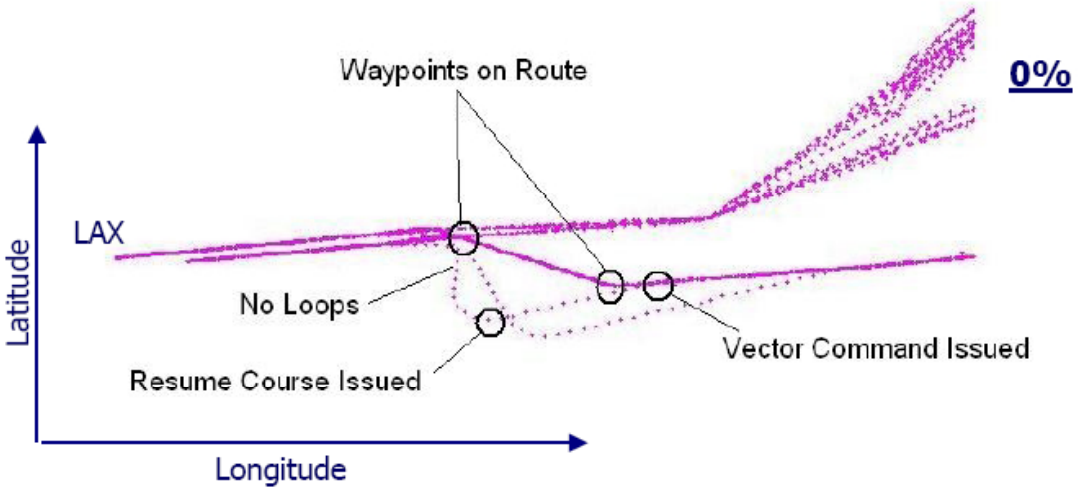

Figure 48: Horizontal profile of arrivals with the changed internal dynamics of the flight management system of aircraft

5.3.3 Summary

The preceding sections illustrate the ease with which a range of design changes may be examined to identify sources of emergent behavior in the models and to make component level transformation to obtain the desired range of behavior (an activity representative of system design and transformation analysis as summarized as analysis 5 in Table 4). The time and effort spent in this exercise was far less compared to the prohibitive

development time found with the previous approach. Using this thesis' conceptual framework and simulation platform, once the basic modeling constructs for the domain had been created, it took one researcher less than one week to test the intuitions, i.e., create new models for component internal dynamics or agent behavior (through the use of facets specific to those behaviors), run a number of simulations to analyze behavior, and feed back the changes in the models. This exercise demonstrates the ease, flexibility and efficiency of using this thesis' conceptual and practical constructs in explaining emergent behavior and testing different design alternatives through the use of component level transformations.

Figure 49 and Figure 50 compare the previous model's performance with that of this thesis' model after correcting the unrealistic emergent behaviors. The graphs show the number of times aircraft got too close to each other ("violations") in each of the five sectors for each of the four scenarios (i.e., different arrival streams in different wind conditions) in both the MIT and the TBM work-processes. The graphs clearly show an improvement in performance as measured by a reduction in the number of violations.

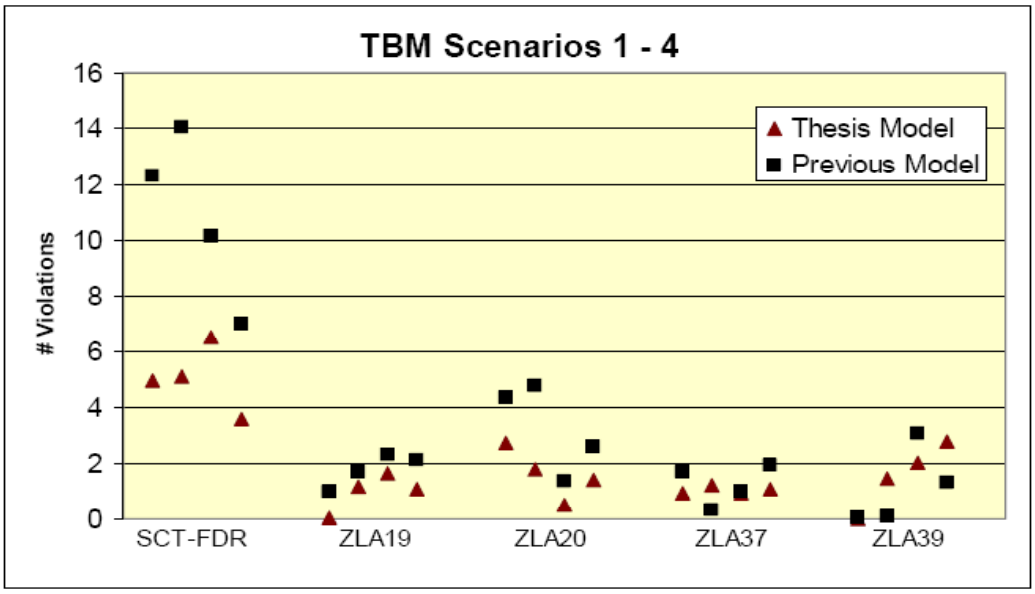

Figure 49: Comparison of average number of separation violations per sector per scenario for previous and corrected TBM models

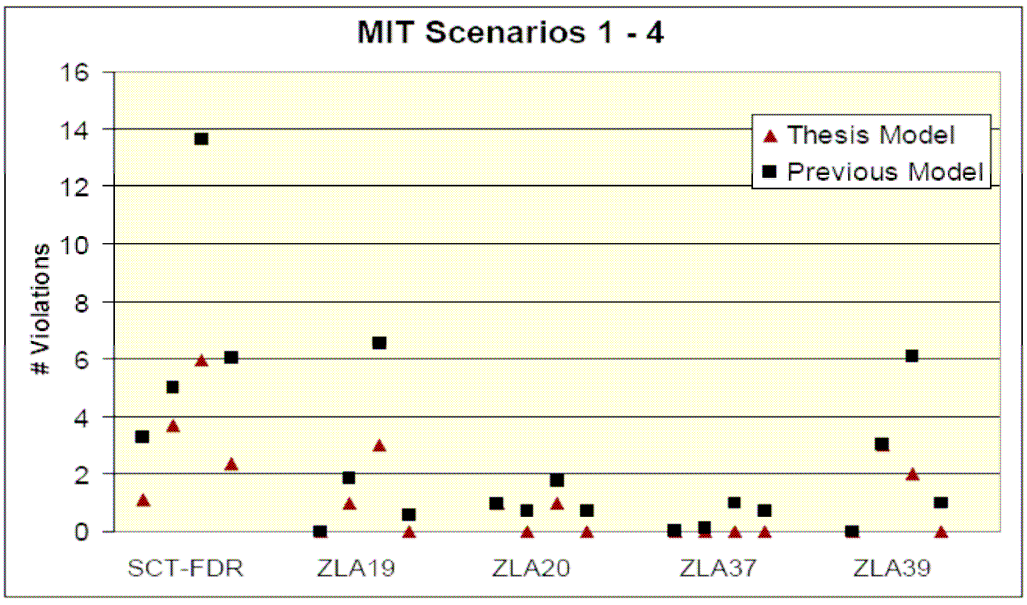

Figure 50: Comparison of average number of separation violations per sector per scenario for previous and corrected MIT models

5.4 Analyzing System Transformations

This section describes two additional analyses of transformations to this air traffic control system. These alternatives are hypothetical and are intended to serve as examples for transformation analysis using this thesis' conceptual framework and simulation platform. The alternatives use two kinds of transformations, one in the work environment and one in the workers, to complement the component level transformations just described in §5.3. A network level transformation in the work environment is exercised through changing the set of work-processes available to each air traffic controller. Transformation in the worker is exercised through changing their skills to be more accurate and have less variation in performance (analysis 3 in Table 4). The performance results of both these transformations are compared with the model of the current system with the unrealistic emergent behaviors removed just described in §5.3.

5.4.1 Comparing Network Level Transformation Alternatives

For the purposing of demonstrating that the modeling and simulation framework can be used to model network level transformations, the system was transformed by changing the work-processes available to each worker, i.e., through changing the configuration of the context of the workers. In the MIT configuration, the controllers of the four higher-altitude sectors (ZLA-39, ZLA-37, ZLA-20 and ZLA-19) use procedures for distance-based separation between aircraft (MIT procedures) which help them issue air traffic clearances to ensure that no two aircraft come closer than the desired in-trail spacing. The air traffic controllers' primary goal was to enforce this MIT restriction and make sure that each aircraft was resumed on its course before it left their sector, if they had earlier

deviated it off course. To date, most simulations had assumed this was the only work-process stipulated to the controllers of these sectors.

In the transformed work environment, each controller was provided with an additional set of work-processes, the conflict avoidance procedures, which are meant to additionally prevent separation violations. To transform the system in this way, the only changes in the system model were (1) to change the goal assignment of the controllers (workers) of the ZLA sectors to also include violation prevention and (2) to add these procedures in the contextual dimension to the context of the ZLA sector controllers. The functional dimension did not need any changes because the procedures already existed in the system for the SCT sector and thus were already associated by means-ends-constraints relationships with the goals. Once these changes were made in the declarative model, i.e., the XML representation of the contextual dimension, the system and agent construction architecture automatically constructed the computational models needed for the simulation. The agent models did not have to change because they were already capable of processing whatever set of work-processes is assigned to them.

Figure 51 compares the performance of the new design alternative with the existing system in terms of the number of separation violations found in each sector over multiple runs in several scenarios. Though performance is improved for the new design alternative, i.e., the average number of violations is reduced, these results could not be compared with reality due to lack of data.

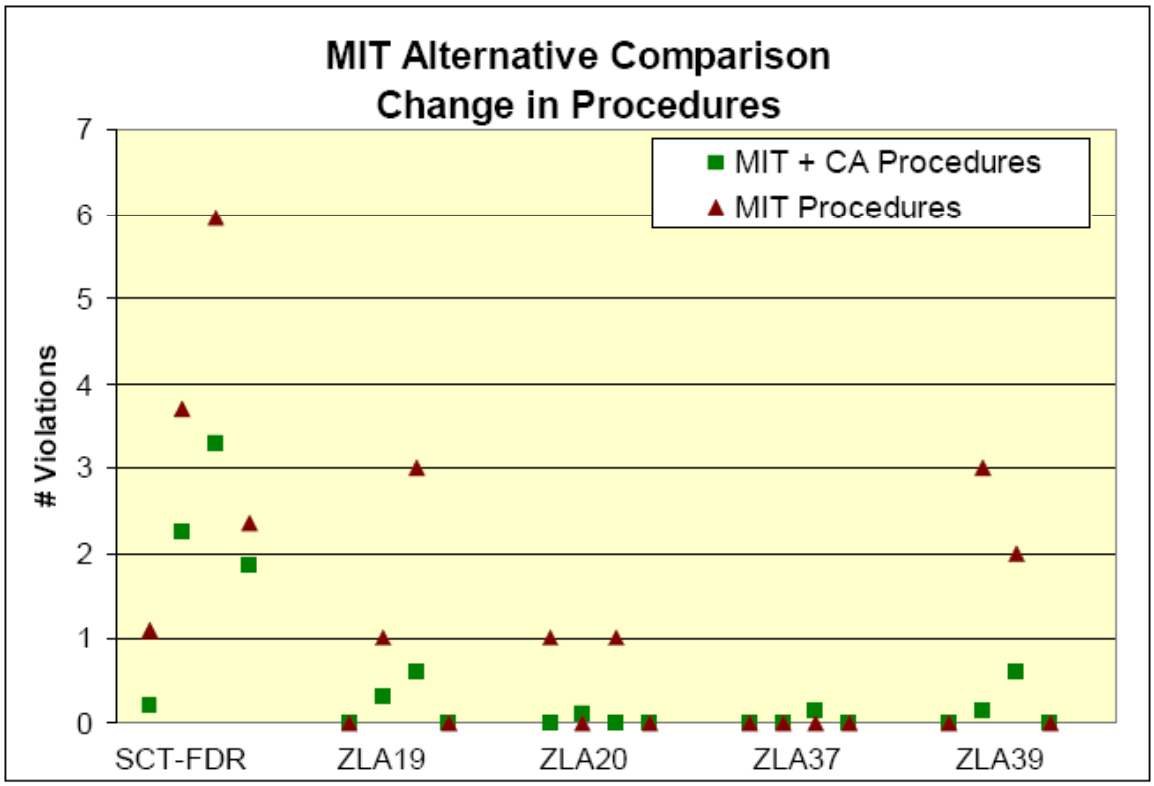

Figure 51: Comparison of average number of separation violations per sector per scenario for changes through the work environment

5.4.2 Comparing Worker Level Transformation Alternatives

This section demonstrates how the work in this thesis can be used to assess worker level transformations. To assess the effectiveness of the procedures by themselves in controlling the workspace, the human performance model was replaced with a worker model with resource limits and without any stochastically induced inaccuracies in its activities in terms of the probability of detecting a conflict or a spacing problem in every scan of the radar display (Table 7). This transformation was enacted using the same

declarative model of the worker by simply (1) replacing the reference in declarative model to the facet for the original resource provider with a modified version that did not limit resources and (2) replaced the reference in the declarative model to the facets for the activities involving inaccuracies to slightly modified facets which did not have inaccuracies. Rerunning the model constructor on the new declarative model of the system created the computational model of the transformed worker and the system.

Table 7: Summary of worker-level transformation in air traffic controller model

Agent Type	**Unlimited resources**	**Limited resources (Max Resources = 7)**		
Source of Variability	Accuracy	Accuracy	# Resources	
Distribution	Uniform	Uniform	Normal	
Activity	Probability	Probability	Mean	Stdev
Monitor Traffic for Conflicts	1	0.8	3	1
Monitor Traffic for MIT Spacing	1	0.8	3	1
Monitor Traffic for TBM Compliance	N/A	N/A	2	1
Change Speed	N/A	N/A	4	1
Change Heading	N/A	N/A	4	1
Change Altitude	N/A	N/A	4	1
Resume Course	N/A	N/A	4	1
Resume Speed	N/A	N/A	4	1
Monitor Sector Boundary Conformance	N/A	N/A	2	1
Wait	N/A	N/A	1	1
Follow Procedures	N/A	N/A	2	1

Simulations for the same scenarios as before yielded the results shown in Figure 52. The improvement in performance (i.e., reduction in violations) was expected due to increased

accuracy and improved response time of the worker. As with the previous transformation this one cannot be validated since the real air traffic system is always operated by resource constrained controllers, but this exercise does demonstrate the ease with which a worker-level transformation can be modeled and examined.

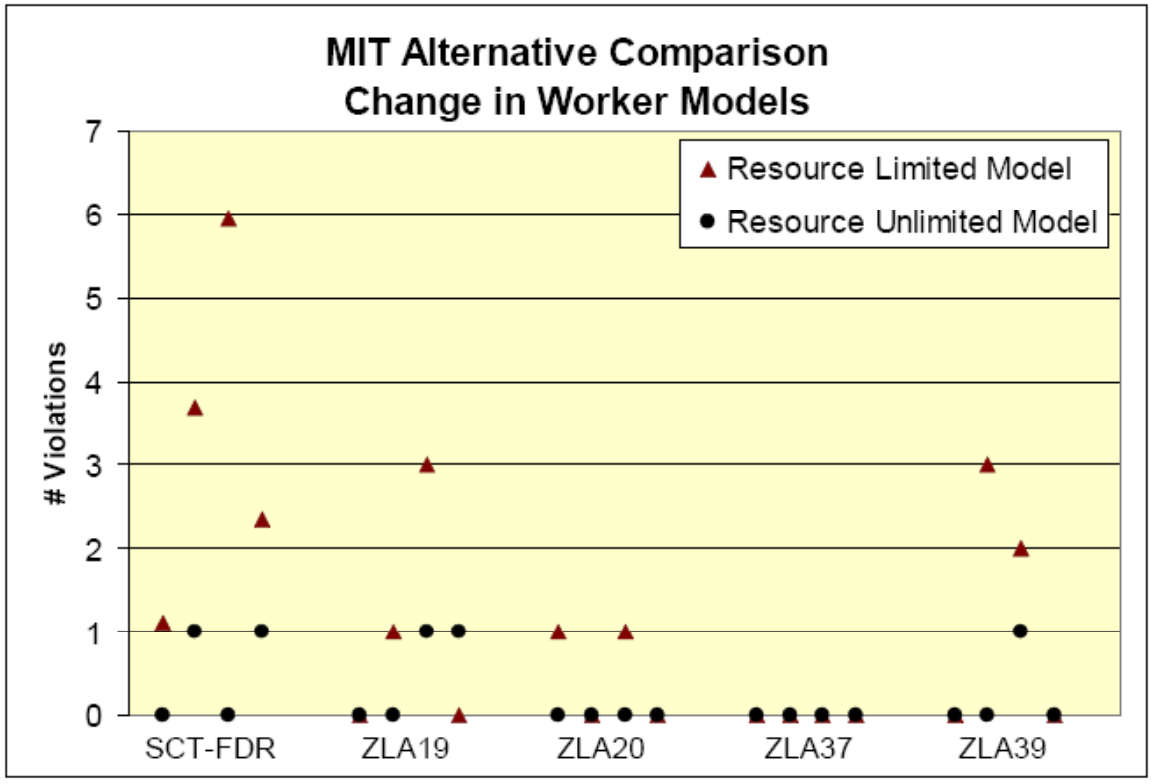

Figure 52: Comparison of average number of separation violations per sector per scenario for changes in worker models

5.4.3 Summary

The two transformations discussed in this section were hypothetical and so could not be validated. Regardless, they serve as good examples to demonstrate the ability to analyze the impact of both network and worker level transformations, both in terms of the ability to model those transformations and in terms of the ease and efficiency gained using this thesis' conceptual framework and simulation platform.

In terms of efficiency it took one researcher about two days to enact the worker level transformation, configure and run 40 simulations for each scenario on each of eight different machines, post process the data, and analyze it. It took about three days to do the same for worker level transformations. This level of effort is sufficiently low as to motivate such analyses as a regular, integral part of many design processes.

5.5 Validating System Models

This section discusses this thesis' efforts to validate the simulation. One of the challenges posed in §5.1 was to create a valid simulation, or to identify causes of discrepancies between simulation behavior and output data that go beyond the scope of the modeling and simulation capability. From the simulation results shown in Figure 53 and Figure 54 we can see that the number of violations in the simulated system is greater than the recorded radar data. In fact, this number of violations would in reality be considered unsafe. To statistically compare the simulation output with observed data from the real system, t-tests were performed between the metrics from the radar data and the simulated system. For more than half of the thirteen metrics collected for each sector the t-tests found significant differences between the outputs from the simulation and the

radar data. Thus validation of the modeled system failed, both for the previous simulation that had unrealistic emergent behaviors in flight profiles and for that developed in this thesis.

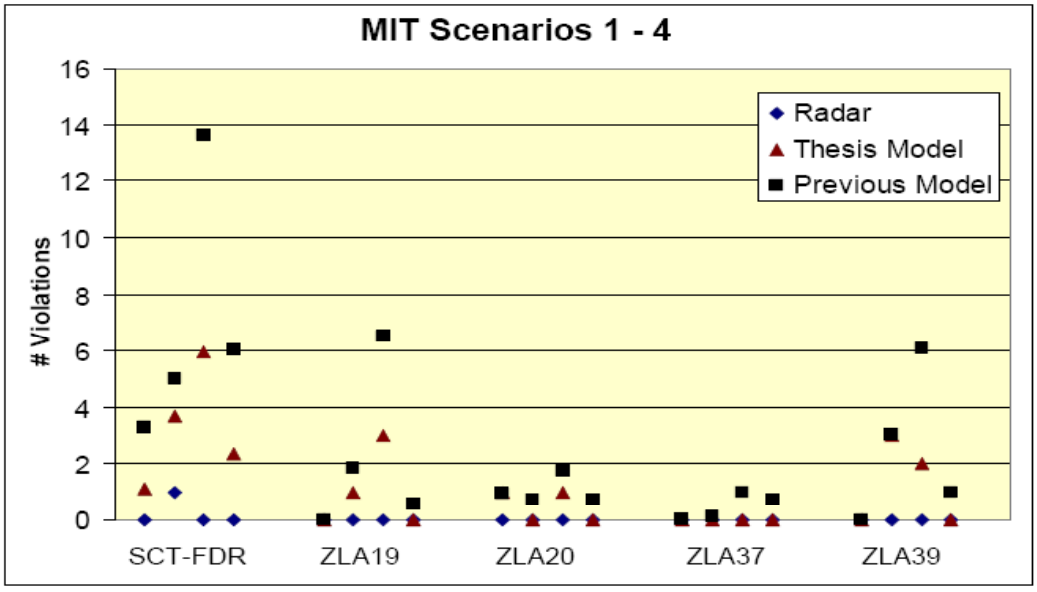

Figure 53: Comparison of average number of separation violations for simulations and the observed radar data for MIT scenarios

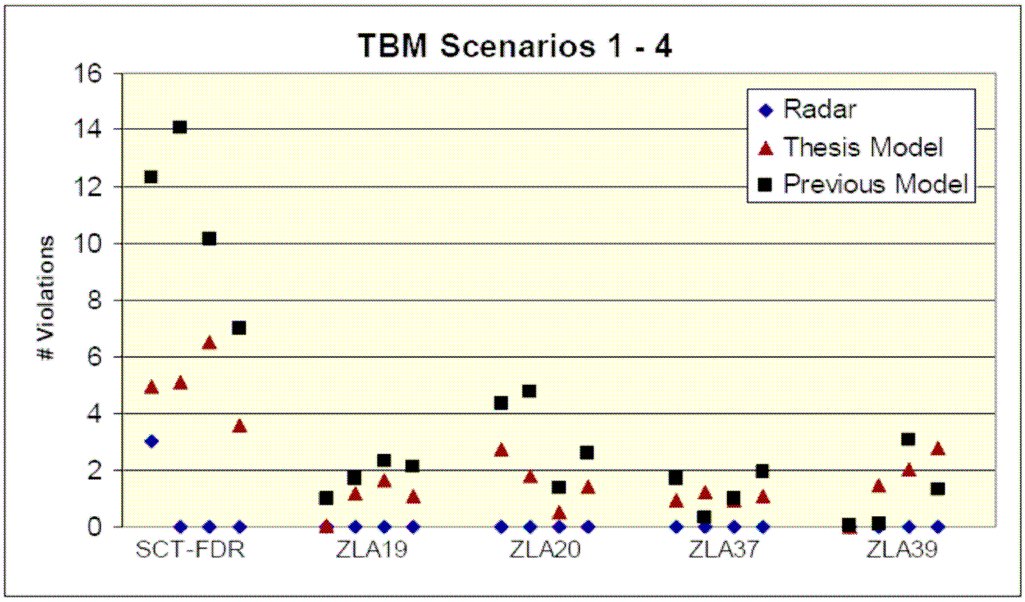

Figure 54: Comparison of average number of separation violations for simulations and the recorded radar data for TBM scenarios

As discussed in §5.1, there could be a number of causes of these failures:

1. *The input data used to configure the simulations could have errors or biases*: Most elements of the input data were crosschecked with a third party and found to be correct. However, with regards to the inputs provided for delay times associated with each aircraft in TBM operations, two problems were found. First, some of the times were sufficiently small that the aircraft could not meet them without an unrealistic increase in speed during a phase of flight where their speed should be reduced in preparation for landing. Second, some of the delay times required aircraft to get sufficiently close that they created separation violations. These delay times were provided by a third party as input data for these simulations. As such, it was impossible for this thesis' research to identify

whether the problems with this input data reflect poor representation by the third party of the output of TMA system that generates them, or whether the TMA system itself is prone to these issues. However, a crosscheck by this third party with radar observations of the real system showed changes in the flight profiles of the aircraft found to have difficulty with delay times in the simulation, suggesting that controllers need to perform more than just the prescribed TBM work-processes for the delay times to be met.

2. *The individual models of the system components may not adequately represent aspects of behavior they were intended to represent*: Each individual component was therefore checked for adequate representation of intended aspects of behavior and was found to be valid for the given specifications. Most of these individual models had been validated during previous simulation developments. For example, the winds calculated by the simulated wind model were compared against radar data through the use of paired t-tests and were found to have similar distributions. The aircraft models were validated in terms of their ability to meet their waypoint restrictions both in terms of their spatial constraints and in terms of speeds, and their speed profiles had been verified. Furthermore, these models had previously been used in other successful research, thus lending further credibility to their correctness. Thus, there was reasonable confidence in the validity of individual models of system components.
3. *The intended aspects of component behavior described by the models may not have included some dynamics critical to emergent system behavior*: A few conversations with subject matter experts revealed that there may have been some

un-modeled dynamics. Specifically, there is a significant level of communication and coordination between the controllers in the use of both MIT and TBM procedures, but this communication had not been specified when preparing the model specifications and therefore had not been modeled. In case of MIT they have letters of agreement between contiguous sectors that specify the exact miles-in-trail restriction which may not be fully represented in the model specifications. In case of TBM there is also evidence of significant coordination and communication amongst controllers to exercise the TBM restrictions (Farley, Foster et al., 2001; Mann, Stevenson et al., 2002). These explorations suggest the likelihood of un-modeled dynamics beyond those the models were intended to cover. Such behavior could be attributed to the creativity of and learning by air traffic controllers, or to commonplace but undocumented practices.

4. *The output data against which the system was being validated could be wrong or biased*: In addition, simulation runs identified some apparent biases in the data about real system behavior provided by a third party for simulation validation. Figure 55 and Figure 56 compare the average distance-in-sector validation measures with those recorded by the simulations. As the graphs show, for MIT there is negative bias in sectors ZLA19 and ZLA20 and positive bias in sectors ZLA37 and ZLA39. The measured values are different by about 12 miles: radar data records a value of approximately 20 miles, but simulated data records a value of nearly 32 miles, an increase of 60%. Examining the profiles of the planned routes (profiles obtained by drawing straight lines through the sectors) it became evident that in reality the aircraft could not have taken such short routes.

Likewise, examining the simulated aircraft profiles, most of the aircraft were not diverted by the controller from their flight plan; the few that were diverted did not divert so far as to increase the average distance of flight of all aircraft by 60%. As a further test, since the number of vectoring commands given in the simulation is significantly small, the average distance-in-sector for flight paths when there is no control by air traffic controllers (i.e., simulating whence the aircraft coast down their flight path) should dominate this measure. As shown in Figure 55 and Figure 56 the simulated data for both the previous and this thesis' simulation is dominated by the no-air-traffic-control scenario as expected. A similar issue manifested in the TBM validation data (Figure 56). Finally, the average total-flight-distance (i.e., the total distance flown by each aircraft through all sectors it traverses during arrival) is almost the same between simulations and validation models, suggesting the radar data intended for validation may have been systematically biased high in sectors ZLA37 and ZLA39, and low in sectors ZLA19 and ZLA20.

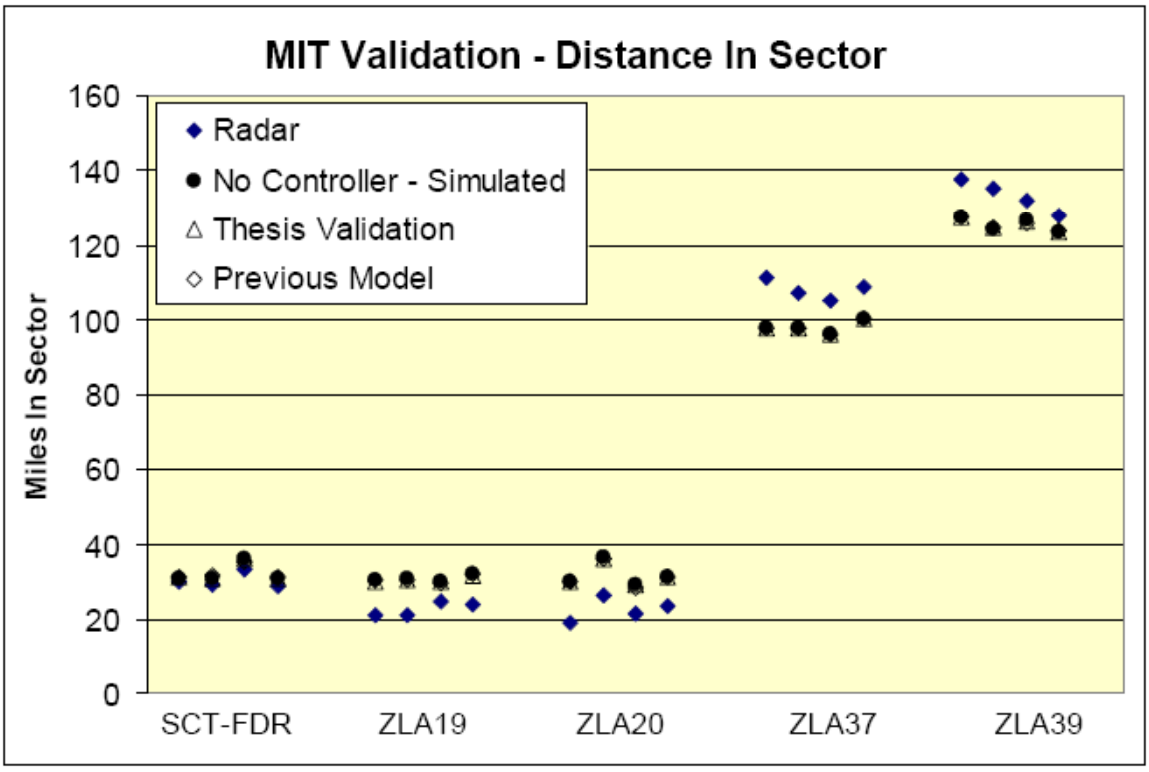

Figure 55: Bias in MIT validation data

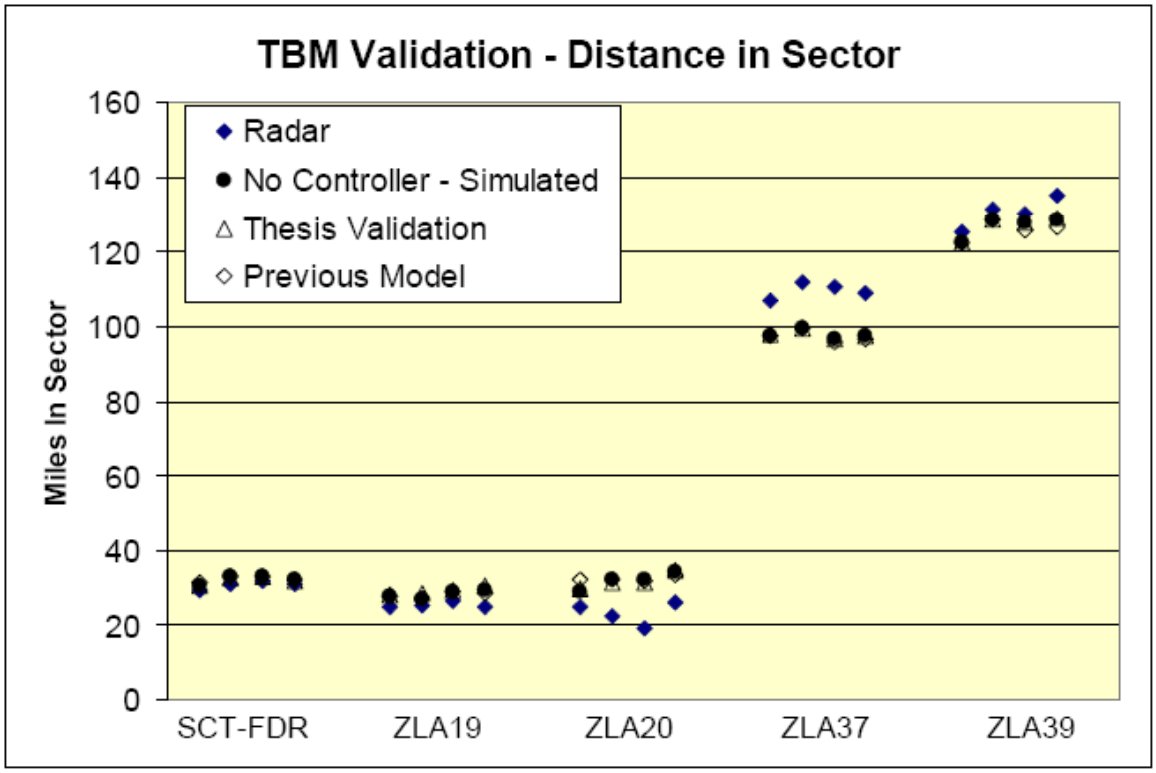

Figure 56: Bias in TBM validation data

5.6 Issues in Engineering Efficiency

This section compares the modeling and simulation efforts in modeling the air traffic control system using this thesis' conceptual framework and simulation platform and the previous simulation. The previous modeling, simulation and analysis effort was spread out over a span of about one and half year, which is significantly large compared to this thesis' three and a half month effort. Table 8 compares the total efforts in modeling and in simulation and analysis. The modeling effort includes: conceptual modeling, development of computational models, troubleshooting, debugging and model

verification. The simulation and analysis effort include: modification of models for testing transformations, configuring simulations, conducting simulations, and analyzing their results.

Table 8: Comparison of estimated modeling and analysis efforts (man-months)

	Previous Simulation	This Thesis
System Modeling	17.5	1.75
Simulation and Analysis	16	1

This thesis' effort definitely required significantly less time and effort, and was able to explore more transformations than the previous effort. I mostly attribute this gain in efficiency to the ability to model the system in a structure-preserving manner, thus being able to make and test transformations much quicker. However, some exogenous factors should be noted. First, there was learning from the first effort that educated this thesis' developments. Second, some of the models and analysis capabilities developed in the previous effort were available for this thesis' effort; attempts to exclude their development time from the time estimates for the previous simulation were conducted in good faith but required subjective assessments.

5.7 Summary

This chapter demonstrated the framework's ability to model socio-technical systems in a structure-preserving manner, explain and predict their emergent behavior, and compare performance of different system design alternatives transformed through component, worker and network level changes. This demonstration employed a case study in modeling and simulating the air traffic control system at the Los Angeles International Airport (LAX). Since the same system had been simulated previously using a different modeling and simulation approach, it served as a benchmark for assessing the utility of this thesis. The previous approach had encountered several challenges in producing a valid simulation. Since the nature of these challenges were related to the ability of transforming and simulating the system model, these challenges served as an effective test-bed for the framework and simulation platform. The challenges could not be tackled in practical time using the previous simulation. This thesis was able to tackle those challenges successfully and in less time. Though valid simulations could not be created in this work, it was shown that it is highly likely that the sources of error in validation arose from factors outside of the scope of the framework's capability to model and simulate socio-technical phenomenon, including potential biases in input data and validation data, and the possibility of un-modeled dynamics.

§5.3 and §5.4 demonstrated that, using this conceptual framework and simulation platform it is possible to transform a system at each of the component, network and worker levels. §5.6 showed that there were significant gains in ease and efficiency in modeling and simulating a socio-technical system using this thesis' conceptual framework and simulation platform. Thus, this chapter substantiates most claims of this

thesis about its modeling abilities and its ability to improve engineering efficiency. However, transformation analysis in this demonstration was limited to operational analysis, as this case study did not provide an opportunity to conduct a network analysis beyond analysis of skill requirements created by automatically assembling the controller's agent model.

CHAPTER 6

CONCLUSION

6.1 Summary

Transformations to socio-technical systems may be enacted through changes in technology, processes, information, workers and organizational strucutres. Typically, these changes are implemented by changes within a component (e.g., changing a technology or procedure) or by changing the interrelation between components (e.g., changing which workers have access to specific information), yet the measures of system performance are not measured at the level of the affected components but instead measured as changes to the overall emergent functioning of all the components taken together. This thesis established a conceptual framework and a simulation platform for modeling and simulation of socio-technical systems for a priori computational analysis of the impact of such transformations. The specific focus of this thesis was on socio-technical systems where the work of the workers and system performance emerging from their collective work are the dynamics that interest the designers and the analysts.

In general, socio-technical systems have certain traits that make their analysis difficult. The performance of the system emerges from localized and discrete interaction of workers and their work environment in a manner that can be difficult to predict. Furthermore, both the work environment and the worker influence each other in

significant ways as the system evolves through time. That is, on the one hand, the structure and dynamics of the work environment affects the choice and outcome of worker activities and, on the other, the structure of the work environment may change as a result of worker activities. In addition to these reciprocal interactions that can create complex system behavior, the behavior of the individual entities can also be non-linear and complex due to interactions within their internal mechanisms. With all these factors put together, it is very hard to explain and predict the system-level impact of behaviors and changes at the level of system components and their interrelations. Thus, it is very hard to analyze the system level impact of transformations as they are usually enacted at the level of these components through the following types of changes:

1. Worker level: changes to the intrinsic characteristics of the workers, such as their skills;
2. Component level: changes intrinsic to the physical, technological and process components of the work environment; and
3. Network level: changes to which components comprise the system and work-relevant relationships between them.

To analyze these transformations, for the purpose of informing design, this thesis developed a conceptual framework that specifies the core principles and constructs to model the system, and a simulation platform that allows for computationally constructing a model of the entire system and simulating it to manifest emergent system performance. In particular, these developments enable two kinds of analysis: First, *operational analysis* examines the evolution of system state through time and the emergence of system performance from work-relevant interactions of the workers and the work environment.

Second, *network analysis* examines the dependencies between the elements of the system, i.e., dependencies that result from the arrangement of work-relevant interrelations between the components of the work environment.

To enable these analyses, the conceptual framework developed in this thesis models both the work environment and the workers in the system and analyzes their work relevant interrelations, both through examining their semantic associations and dependencies and through simulation of their collective dynamics through time. This thesis builds on the principles of cognitive engineering to describe the components of the work environment, i.e., technology, processes and information, in work relevant ways and using a structure-preserving model, i.e., a model form that describes their aspects using the same attributes and structure as used by system designers and operators. This thesis also builds on the principles of agent-based modeling to model workers and their interactions with the work environment. These models are specified through declarative models describing which components are included within the system and their interrelations, and computational models of those complex, dynamic behaviors that cannot be adequately described declaratively.

Declarative modeling enables easy composition and modification of component models; in addition by computationally assembling all required components and their interrelations collectively, enables automatic generation of a declarative model of the system, which can be analyzed for network dependencies, i.e., also enables network analysis. To facilitate declarative modeling and network analysis, this thesis both established an XML representation for the declarative models and developed a mechanism that then automatically assembles, from the individual components'

specifications and interrelations, a network-level model of the entire system in XML which can serve to analyze network dependencies between components.

Likewise, this thesis developed a software architecture in which complex, dynamic internal behaviors of components and workers can each be encapsulated as object-oriented computational objects using any of a wide range of model structures as appropriate. The combination of the declarative and object-oriented models also enables computational simulations to predict the system performance that will emerge from a network of components when placed in a given scenario. Thus, this thesis also developed an agent-based simulation platform that can simulate the collections of worker and work environment models created using the conceptual framework for the purpose of operational analysis.

In developing the simulation platform, this thesis employed and integrated a number of advanced principles in software engineering to devise an architecture that relates declarative and computational model specifications in semantically correct ways, and provides for modular construction of component and system models.

The theoretical and practical utility of this thesis' developments was demonstrated through a case study in air traffic control. This case study presented a real-world modeling and analysis problem for analyzing component and network level transformations in work processes in the work environment of air traffic controllers. Known to be a significantly sized and complex analysis problem that had been previously examined using a different agent-based simulation, this case study served to both substantiate this thesis' work at a theoretical level, and to compare its performance with the previous effort to demonstrate its practical benefits. The case study demonstrated the

ability to model and analyze transformations at each of the component, worker and network levels, and to explain and predict emergent performance. The practical benefits included an order of magnitude gain in efficiency of performing the analysis. By making the level of effort feasible within a design process, this thesis also demonstrated its effectiveness in handling complex analyses.

An isolated network analysis was shown for an isolated model; it is my claim, based on first principles, that given the opportunity the computational framework and simulation platform presented in this thesis are capable of performing more substantial network analysis spanning broader sections of socio-technical systems.

6.2 Contributions

This thesis makes several contributions to the fields of cognitive engineering, agent-based modeling and simulation, and system design and analysis:

1. The primary contribution of this thesis is the provision of a conceptual framework to model and analyze transformations that are made at the level of workers, components and their networks. This framework is inherently structure-preserving and computational, thus enabling efficient design analysis through the use of simulations. This framework identifies:
    a. Which elements in the system and within the transformation variables need to be modeled; and
    b. How these elements fit together to model the work environment and the worker models.

Furthermore, this framework is designed to enable operational and network analysis.

2. In identifying the relevant model elements, this thesis extends current cognitive engineering models, which typically only include the physical components of the work environment, by also modeling the non-physical components such as work-processes. These models allow for a more comprehensive view of socio-technical systems.
3. Unlike current cognitive engineering models that model only the means-ends and parts-whole relationships between the physical components, this thesis enables modeling the work environment through a variety of work-relevant relationships, again providing for a more comprehensive view of socio-technical systems.
4. This thesis provides for two kinds of transformation analysis: 1) operational analysis that evaluates the impact on emergent system performance through simulations and 2) network analysis that identifies dependencies between system components through declarative specification and querying of their collective models and their interrelations.
5. This thesis constructs a domain-independent software architecture and simulation platform to computationally analyze the impact of transformations. This platform is a tool for practitioners and researchers in system design to test their intuitions about enacting system transformations through component, worker and network level changes. This architecture is built on the conceptual framework to establish a structure-preserving and design-driven approach to constructing system models,

thus significantly reducing the complexity of constructing and analyzing system models.

6.3 Future Directions

1. Even though this work demonstrated the conceptual framework as being useful for socio-technical systems through the use of one case study, this thesis' work would need to be tried in a variety of domains and with a variety of design transformation problems before it can be widely accepted as a useful way of looking at system transformations. Such wide spread application is important to thoroughly assess the utility of the framework in modeling systems and its effectiveness in easing the job of designers and analysts, and to better identify the class of socio-technical systems it is best suited for.
2. Any concrete and systematic approach helps achieve efficiency and thoroughness in modeling and analysis. However, if the approach is accompanied by tools that enable visual modeling and analysis, it significantly improves the practitioner's efficiency, reduces the possibilities of human error and reduces the semantic gap between conceptual formulations and mental models of the modelers. Furthermore, if domain specific repositories of component models are created and made available the modeling effort for subsequent analyses becomes small. There is a need of such practical developments for this thesis' framework. Availability of such tools is also likely to improve acceptance of this thesis' conceptual framework and simulation platform and models.

3. Network analysis, as discussed in this thesis, addresses network-level dependencies within snapshots of the system at any particular point in time. However, similar to the emergence and evolution of system behavior and performance from the interactions of the "low" level components, network level dependencies also evolve through time and may change due to discrete and localized interactions of the workers and the work environment. Analysis of this evolution should provide useful insights for informing the design of transformations, and requires further research for development of models and approaches that allow analysis of the dynamism of network dependencies.
4. This framework and simulation platform was developed with the intention of supporting designers in analyzing transformations enacted in terms of changes to components, their relationships and their organization. I characterize these changes as 'low' level, yet system performance emerges from them. However, there are higher levels of abstraction that are also used when designing changes in a system: while they are enacted through 'low' level changes, system designers often start thinking in terms of changes at the system level or some 'middle' level of abstraction. An example from this thesis' test case is: what would happen if one could achieve a specific minimum separation between aircraft approaching the airport? Through the use of mathematical models one can find that such a transformation will improve efficiency. However, the problem remains about how one should enact this change at the 'low' level. I believe that theoretical or heuristics-based developments are needed to guide the designers in making these translations between upper, middle and lower levels. Once such guidance is

available about which ‘low’ level transformations should be tried, the designer can use the developments in this work to cross check if those ‘low’ level transformations actually work with the inherent limitations in the workers or the limitations of available technology. For example, one could think of a process that that automatically tries several design alternatives produced by perturbing the design variables through the use of some stochastic or heuristic-based hill-climbing algorithm. I believe research in such a direction is a natural successor of this work.

APPENDIX A

CASE STUDY: COMPUTATIONAL MODEL SPECIFICATIONS

This appendix presents the model details for the environmental components and the agents for the case study discussed in Chapter 5 of this thesis. There was one kind of agent, the air traffic controller. There were five instances of this agent type, which differed slightly in their skill set based on the configuration of their workspaces, each of which was different from the rest in terms of the work-processes comprising the workspace. The system consisted of the following types of environmental components:

1. Aircraft: Each simulated scenario had approximately 125 instances of this component.
2. Sector: There were five sector components representing ZLA-39, ZLA-37, ZLA-20, ZLA-19 and SCT-FDR.
3. Radar Equipment: Each sector had one radar equipment assigned to it. This component provided the radar screen in the context of the workspace of the controller.
4. Voice Radio: Each sector had one voice radio assigned to it. This component provided the sector's communication channel in the context of the workspace of the controller.
5. Flight Strips: Each sector had one of these components assigned to it. This component represented a collection of all flight strips listing the flight plan of each aircraft in the sector.
6. Work-process: There were a number of instances of this type, each representing a distinct air traffic control procedure. Each sector had assigned to it a number of these work-processes.

Description of the Contextual Dimension

A representative illustration of the contextual dimension was shown in Figure 40 in Chapter 5. The structure of this dimension for each controller is reasonably simple: the contextual dimension subsumes each of the five sector components; each of those subsumes the radar screen aspect of the radar equipment, the sector channel aspect of the voice radio equipment, the flight plans aspect of the flight strips component, and all the work-processes assigned to that sector; each radar screen subsumes the radar data aspect of the aircraft in its sector; each sector channel subsumes the command mechanisms aspect of the aircraft in the sector; and finally each flight plans aspect subsumes the fight plan aspect of each aircraft in the sector. Since, each controller has access to the sector component's top-level contextual node, he or she can access all contextual nodes subsumed in the hierarchy just discussed. However, as may have been noted, this does not include contextual nodes of those aircraft outside their own sectors.

Description of the Component Models

This section describes each of the components in terms of:

1. The facets that make up the component and how they build on each other
2. The internal dynamics
3. The Aspects it provided for any dimension

Aircraft

Facets

Table A1 lists and describes the facets of the aircraft component. The numbers in the braces list the facets on which a particular facet builds.

Table 9: Facets of the Aircraft component

SN	Facet	Description
1	BasicAircraftData	Stores the state of the aircraft, provides the state properties for the contextual dimension.
2	BasicFlightPlan	Stores the flight plan of the aircraft, provides the flight plan to be read out in the contextual dimension.
3	BasicFlightProcessor (1)	Provides the basic flight dynamics to the aircraft.
4	BasicFMSProcessor (1, 2)	Provides the waypoint following capability to the aircraft.
5	ChangeAltWithFPA (1)	Provides the aircraft with the ability to be commanded to change altitude with a given flight path angle (usage-mechanism).
6	ChangeAltWithVS (1)	Provides the aircraft with the ability to be commanded to change altitude with give vertical speed (usage-mechanism).
7	ChangeSpeed (1)	Provides the aircraft with the ability to be commanded to change speed to given speed (usage-mechanism).
8	FlyToHeading (1)	Provides the aircraft with the ability to be commanded to change heading to given heading (usage-mechanism).
9	IFFDataRecorder (1)	Metrics collector that collects IFF data for the given aircraft and logs it to the simulation runs IFF file. IFF data is a proprietary format used by a third party to aircraft record track points and events in the simulation.
10	ResumeAltitude (1, 2)	Provides the aircraft with the ability to be commanded to resume waypoint altitude following, without resuming 2D course or waypoint speed (usage-mechanism).

Table 9 (continued)

11	ResumeCourse (1, 2)	Provides the aircraft with the ability to be commanded to resume 3D course following (usage-mechanism).
12	ResumeSpeed (1, 2)	Provides the aircraft with the ability to be commanded to resume waypoint speed restrictions (usage-mechanism).
13	RadarData	A contextual node. Maps several state properties onto the contextual dimension. This aspect expects the properties to be available irrespective of which facet provides it. It does not, in principle, build up on any facet, but in this model it requires BasicAircraftData.
14	SectorChannel	A contextual node. Maps five usage mechanisms on the contextual dimension. This aspect expects the usage-mechanisms to be available irrespective of which facet provides it. It does not, in principle, build up on any facet, but in this model it requires ChangeAltWithVS, ResumeSpeed, ResumeCourse, FlyToHeading and ChangeSpeed facets to be available.
15	FlightPlan	A contextual node. Maps the state of the flight plan property onto the contextual dimension. This aspect expects the properties to be available irrespective of which facet provides it. It does not, in principle, build up on any facet, but in this model it requires BasicFlightPlan to be available.

Internal Dynamics

There are two processors that run in parallel to accomplish the basic flight characteristics and the waypoint following dynamics of the aircraft.

1. BasicFlightProcessor: This processor manages the basic flight mechanics of the aircraft with the use of a 3DOF model. This processor works with the state of the aircraft that is stored in the BasicAircraftData facet.

2. BasicFMSProcessor: This processor manages the waypoint following dynamics of the aircraft. This processor works with the state of the aircraft that is stored in the BasicAircraftData facet.

Aspects

1. RadarData: provides the Latitude, Longitude, Altitude, GroundSpeed, VerticalSpeed, Heading, and FlightPathAngle properties to the contextual dimension.
2. FlightPlan: provide the FlightPlan property to the contextual dimension.
3. SectorChannel: provides the following usage mechanisms to the contextual dimension. The name in the braces lists the facet that provides this usage mechanism:
    a. changeSpeed (ChangeSpeed): Command the aircraft to change speed to given speed.
    b. changeAltWithVS (ChangeAltWithVS): Command the aircraft change target altitude to given altitude with given vertical speed.
    c. flyToHeading (FlyToHeading): Command the aircraft to fly to the given true heading
    d. resumeCourse (ResumeCourse): Command the aircraft to resume course, i.e., flight plan
    e. resumeSpeed (ResumeSpeed): Command the aircraft to resume waypoint speeds.

Radar Equipment

Facets

Table 10: Facets of the Radar Equipment

SN	Facet	Description
1	AirspaceScanner	
2	RadarScreen	contextual node that subsumes the RadarData contextual nodes of all the aircraft in the sector. This contextual node is subsumed by the contextual node that represents the context of the air traffic controller.

Internal Dynamics

1. AirspaceScanner: Scans the airspace for all aircraft that are in the sector to which a particular radar is assigned, and includes their radar contextual nodes under its own contextual node. The aircraft ID is also added to the list of aircraft, a property maintained and exposed by this processor.

Aspects

1. RadarScreen: provides a list of all aircraft currently in the sector to the contextual dimension. Subsumes all RadarData contextual nodes of all aircraft in sector.

Voice Radio

Facets

Table 11: Facets of the Voice Radio component

SN	Facet	Description
1	AirspaceScanner	
2	SectorChannel	Contextual node that subsumes the SectorChannel contextual nodes of all the aircraft in the sector. This contextual node is subsumed by the contextual node that represents the context of the air traffic controller.

Internal Dynamics

1. AirspaceScanner: Scans the airspace for all aircraft that are in the sector to which a particular radar is assigned, and includes their radar contextual nodes under its own contextual node.

Aspects

1. SectorChannel: Subsumes SectorChannel contextual nodes of all aircraft in sector

Flight Strips

Facets

Table 12: Facets of the Flight Strips component

SN	Facet	Description
1	AirspaceScanner	
2	FlightStrips	Contextual node that subsumes the FlightPlan contextual nodes of all the aircraft in the sector. This contextual node is subsumed by the contextual node that represents the context of the air traffic controller.

Internal Dynamics

1. AirspaceScanner: Scans the airspace for all aircraft that are in the sector to which a particular radar is assigned, and includes their radar contextual nodes under its own contextual node.

Aspects

1. FlightStrips: Subsumes FlightPlan contextual nodes of all aircraft in sector and puts them on the contextual dimension.

Sector

The sector component was static, i.e., it did not have any internal dynamics. It was developed only to represent the workspace of a controller.

Facets

Table 13: Facets of the Sector component

SN	Facet	Description
1	Sector	Contextual node that subsumes the contextual nodes of all components in the workspace of a controller.

Aspects

1. Sector: Subsumes the contextual nodes of all components in the workspace of a controller and puts them on the contextual dimension.

Work-Process

A work-process is a static component and does not have any internal dynamics. But it provides one aspect to expose its process.

Table 14: Facets of the Work-process component

SN	Facet	Description
1	WorkProcess	Contextual node that exposes the expression of the situation in which the work-process is applicable, and the underlying process

Aspects

1. WorkProcess: Exposes the work-process to the contextual dimension.

Description of Worker Models

There was only one kind of worker, the air traffic controller. This description lists the complete set of skills and facets that was used for the air traffic controller models.

Air Traffic Controller

Processors

1. ActivityProcessor: Runs all the activities that are started by other activities. These include activities such as monitoring traffic in the airspace, and other activities that are started by the procedure processor.
2. ProcedureProcessor: Processes the currently running procedures.

Skills

This section lists and briefly describes the skills aggregated in the air traffic controller. These skills are listed under the heading of the facet that brings these skills into the air traffic controller model.

Table 15: Skills of the Air Traffic Controller listed with the facet that provides those skills to the agent model

Facet		
	Skill	**Description**
BasicAnalyticalCapabilities		
	AreACInConflict	checks if two aircraft are in conflict
	GetIfCommonWaypoint	Checks if two aircraft have a common waypoint
	GetDistanceBtPts	get horizontal distance between two points
	GetHdgBtPts	gets heading between two waypoints, as measured from the North.
	GetIfFailedResolution	makes sure if a particular resolution failed for given aircraft in a given conflict.
	IsPointInSector	checks if a given point is in sector
	GetDstncToMergePoint	Calculates the distance between an aircraft's current position and the merge point of that aircraft and another aircraft.
	GetHdgFromMergePoint	Gets the orientation of an aircraft from the merge point of that aircraft and another aircraft.
BasicExecutionCapabilities		
	ChangeSpeed	Command a given aircraft to change speed.
	ChangeAltWithVS	Command a given aircraft to change altitude with a given vertical speed.
	ChangeHeading	Command a given aircraft to change heading to a given heading
	ResumeCourse	Command an aircraft to resume course
	ResumeSpeed	Command an aircraft to resume waypoint speed
	ResumeAltitude	Command an aircraft to meet waypoint altitude restriction
	Wait	Wait for a given time
BasicPerceptualCapabilities		
	GetDoubleProperty	read a variable from the context
	GetFlightPlan	read flight plan of a given aircraft from the context
CmonitorConformance		
	MonitorConformance	monitor boundary conformance for all aircraft sent off-course by the controller
CmonitorTraffic		

Table 15 (continued)

	MonitorTraffic	monitor traffic in sector for possible conflicts.
CResolutionCheckingCababilities		
	IsChangeConflictFree	judge if a particular maneuver for a particular aircraft would be conflict free in given time frame
	CalculateLatLongAltAtTimeAhead	Calculate the position of an aircraft in given future time.
	IsACReadyToResumeAlt	make sure that the aircraft can resume altitude without creating future conflicts.
	IsACReadyToResumeSpeed	make sure that the aircraft can resume speed without creating future conflicts.
	IsACReadyToResumeHeading	make sure that the aircraft can resume course in two dimensions (not vertically) without creating future conflicts.
	IsACReadyToResumeCourse	make sure that the aircraft can resume three-dimensional course without creating future conflicts.
	IsACOnAltChange	check if the aircraft has been commanded altitude changes.
	IsACOnHeadingChange	check if the aircraft has been sent off-course.
	IsACOnSpeedChange	check if the aircraft has been commanded to change speed.
MITResolutionCheckingCapabilities		
	MonitorMITTraffic	monitor arrivals for in-trail spacing violations
	IsACInMITViolation	check if a particular aircraft is closer to any other arriving aircraft than the required in-trail spacing.
	GetViolationDistance	Calculate the distance that a particular aircraft will have to absorb to avoid in-trail separation violation.
	GetDistanceToDestination	Calculate aircrafts along track distance from destination
	GetHeadingFromDestination	Calculate orientation of a given aircraft from the destination.
	IsPastILSMerge	Calculate if an arrival aircraft is past the ILS merge point for the destination.
	GetHeadingFromILSMerge	Calculate the orientation of the aircraft from the ILS merge
	GetDistanceToILSMerge	Calculate the along track distance of an aircraft from the ILS merge point

Table 15 (continued)

CgetConflitTimeAhead		
	GetConflictTimeAhead	Calculate time to conflict
ProcedureProcessor		
	FollowProcedures	follow the procedures in the context

Capabilities

1. ATCData: stores the internal state of the air traffic controller
2. ResourceProvider: provides a pool of unitary internal resources for a resource limited agent. An activity requesting resources cannot start unless the required number of resources are available on the resource provider. If a high priority activity requests resources, the resource provider suspends low priority activities if they or a combination of them can provide the required number of resources.

Facets

The numbers in the braces list which facet the listed facet builds on.

Table 16: Facets of the Air Traffic Controller agent

SN	**Facet**	**Description**
1	ATCData	Refer preceding sections
2	ResourceProvider	Refer preceding sections
3	ContextAccessor	Provides access to the context of the air traffic controller
4	ActivityProcessor	Refer preceding sections

Table 16 (continued)

5	ProcedureProcessor (3)	Provides a general mechanism for processing procedures. Thought the procedure processor itself doesn't depend on other aspects, the procedures may require skills that come from other aspects.
6	BasicPerceptualCapabilities (3)	Refer preceding sections
7	BasicAnalyticalCapabilities (1, 6)	Refer preceding sections
8	BasicExecutionCapabilities (1, 2, 3, 4, 5, 7)	Refer preceding sections
9	CResolutionCheckingCababilities (1, 6, 7)	Refer preceding sections
10	CmonitorTraffic (1, 2, 3, 4, 6, 7, 9)	Refer preceding sections
11	CmonitorConformance (1, 2, 3, 4, 7, 9)	Refer preceding sections
12	CgetConflitTimeAhead (1, 7, 9)	
13	MITResolutionCapabilities (1, 2, 3, 4, 6, 7, 8, 9)	Only needed when the air traffic controller is following MIT procedures
14	TBMMonitoringCapabilities (1, 2, 3, 4, 7, 8, 9)	Only needed when the air traffic controller is following TBM procedures

APPENDIX B

GLOSSARY

Aspect: A grouping of attributes of an environmental component that represent its dimension relevant view in the multidimensional model of the work environment.

Computational Models: Representations of abstractions capable of being used by computer programs.

Conceptual Framework: A specification of core principles and constructs specified to the level of detail that enable computational modeling of the socio-technical system and its elements.

Conceptual Models: Natural language or equation-based abstractions meant to explain and predict certain kinds of phenomenon in certain kinds of domains.

Contextual Dimension: This dimension defines the structure of a worker's workspace formed by interrelations that identify some system component or their attributes being available in the context of the other

Declarative Models: Models that computationally specify entities through their structural attributes, and through semantic associations with other entities.

Environmental Component: Any physical or non-physical element of the system that affects the choice and/or outcome of worker activities.

Facet: A facet constitutes a set of computational functions and resources, i.e., algorithms and state variables as represented in object-oriented programs. Facets are the fundamental units of constructing an object.

Functional Dimension: A structure that is formed by means-ends-constraints relations between system components.

Knowledge Dimension: An arrangement of the parts of the work environment in some specific work-relevant manner.

Means-Ends-Constraints Relations: An extension of means-ends relation, which identifies a component as being either a means or a constraint to achieving a goal.

Means-Ends Relations: Those work-relevant relations between a system component and a system or individual goal that identify the component as being a means to achieving the goal. One should note that traditional definitions of means-ends relations as used in cognitive engineering identify this relationship between any two components, where neither component has to be a goal.

Network: In this thesis, a network is formed from work-relevant interrelations between system components.

Network Analysis: The act of analyzing the interrelations between and arrangement of the parts of a system. Such an analysis can identify the dependencies between the parts of the system.

Object: A basic computational abstraction in this thesis that is used to represent any individual entity at any level of granularity. An object can be composed of other parts, it can be a specialization of other objects, it can be interrelated with others in networks, and it can interact with other objects.

Operational Analysis: The act of analyzing the evolution of system state through time. In socio-technical systems, system performance emerges from worker behavior, the state of their environment, and their interactions. Thus, in this thesis

operational analysis is conducted by simulating these elements' collective behavior through time and observing the system state that emerges.

Simulation: A simulation is an imitation of some real thing, state of affairs, or process. In the context of this thesis, a simulation is an evolution of system model's state through time.

Simulation Platform: A software and computational specifications of the main modules and mechanisms underlying the conceptual framework. When implementing models, one uses the platform to assure that models adhere to the underlying conceptual framework. The platform provides mechanisms to simulate the modeled system.

Socio-technical System: A system whose elements include workers and their work environment, which itself is comprised of components such as technologies, processes and information.

Structure: An interrelation or arrangement of parts

Structure-preserving Models: Models describing the system, its components and their interrelations using the same attributes as used by system designers and operators.

System: A meaningful group of interacting, interrelated, or interdependent elements forming a complex whole. This group of elements may be arranged in one or more structures, where each element is a part and each structure is defined through the use of a disjoint or overlapping set of structural attributes on each part and through specific kind of interrelations.

Work: All purposeful activities, physical or cognitive, intended towards achieving a specific goal(s).

Worker: The definition of a worker is based on the definition of work as purposeful activity, intended towards the worker's indirect goals, system goals or both.

Work environment: A grouping of all environmental components and their work-relevant interrelations.

REFERENCES

Anderson, J. R. (1993). Rules of the mind, Lawrence Erlbaum Associates.

Anderson, J. R. and C. Lebiere (1998). The Atomic Components of Thought. Mahwah, NJ, Lawrence Erlbaum Associates.

Arkin, R. A. (1998). Behavior Based Robotics, The MIT Press.

Bargiela, A. (2000). Strategic direction for simulation and modeling. Plenary Session, Summer Computer Simulation Conference, Vancouver.

Benda, P. J. and P. M. Sanderson (1998). Towards a dynamical model of adaptation to technological change. OzCHI 1998, Adelaide, South Australia.

Boag, S., D. Chamberlin, et al. (2005). "XQuery 1.0: An XML Query Language."

Bratman, M. E. (1987). Intentions, plans and practical reason. Cambridge, MA, Harvard University Press.

Bratman, M. E., D. J. Israel, et al. (1988). "Plans and resource-bounded practical reasoning." Computational Intelligence 4. pp 349-355.

Bray, T., J. Paoli, et al. (2004). "Extensible Markup Language (XML) 1.0."

Brooks, R. A. (1986). "A robust layered control system for a mobile robot." IEEE Journal of Robotics and Automation 2(1). pp 14-23.

Brooks, R. A. (1991). Intelligence without reason. 12th International Joint Conference on Artificial Intelligence (IJCAI-91), Sydney, Australia.

Brown, A. W., Ed. (1996). Component-Based Software Engineering: Selected papers from the software engineering institute, Wiley-IEEE Computer Society Press.

Bruns, G., P. Mosinger, et al. (2003). XRaptor: A simulation environment for continuous virtual multiagent systems. Available: http://www.informatik.uni-mainz.de/~polani/XRaptor/XRaptor_7_3_8a/xr-uman-screen-7.3.8a.pdf. Accessed: April 2005

Byrne, M. D. (2001). "ACT-R/PM and Menu Selection: Applying a cognitive architecture to HCI." International Journal of Human-Computer Studeis 55. pp 41-84.

Byrne, M. D. (2003). Cognitive Architecture. J. A. Jacko and A. Sears (Eds.). The human-computer interaction handbook: Fundamentals, evolving technologies and emerging applications. Mahwah, NJ, Lawrence Erlbaum Associates. Human Factors and Ergonomics. pp 97-117.

Cantone, D., E. Omodeo, et al. (2001). Set theoary for computing: from decision procedures to declarative programming with sets. New York, NY, USA, Springer.

Castelfranchi, C. (2000). Engineering social order. ESAW00, Berlin.

Clark, J. and S. DeRose (1999). XML Path Language (XPath). Available: http://www.w3.org/TR/xpath. Accessed: March 2005

CMU-SEI (2004). Predictability By Construction: Building high stakes systems from certified software components. Available: http://www.sei.cmu.edu/pacc/files/pacc.pdf. Accessed: May 2005

Codd, E. F. (1970). "A relational model of data for large shared data banks." Communications of the ACM 13(6). pp 377-387.

Corker, K. M. (1994). Man-machine integration design and analysis system (MIDAS) applied to a computer-based procedure-aiding system. 38th Annual Meeting of the Human Factors and Ergonomics Society, Santa Monica, CA, Human Factors and Ergonomics Society.

Decker, K. S. (1994). Environment centered analysis and design of coordination mechanisms. Department of Computer Science. Amherst, University of Massachusetts. PhD.

Decker, K. S. (1998). Task environment centered simulation. M. Prietula, K. M. Carley and L. Gasser (Eds.). Simulating Organizations: Computational models of institutions and groups, AAAI Press/MIT Press.

Decker, K. S. and V. Lesser (1994). The environment centered design of organizations. Autonomous Agents and Artificial Intelligence.

Durfee, E. H. (2000). Distributed Problem Solving and Planning. G. Weiss (Ed.). Multiagent systems: A modern approach to distributed artificial intelligence. Cambridge, MA, The MIT Press. pp 121-164.

Eggleston, R. G. (2002). Cognitive Systems Engineering at 20-something: Where Do We Stand? M. D. McNeese and M. A. Vidulich (Eds.). Cognitive systems engineering in military aviation environments: avoiding cogminutia fragmentosa! Human Systems Information Analysis Center. pp 15-78.

FAA (2005). National Airspace System Operational Evolution Plan 2005-2015: Executive Summary. Available: http://www.faa.gov/programs/oep/v7/Executive%20Summary/Executive%20Summary%20v7.pdf. Accessed: April 2005

Fallside, D. C. and P. Walmsley (2004). XML Schema Part 0: Primer Second Edition. Available: http://www.w3.org/TR/xmlschema-0/. Accessed: June 2005

Farley, T., J. D. Foster, et al. (2001). A Timd-Based Approach to Metering Arrival Traffic to Philadelphia. First AIAA Aircraft Technology, Integration, and Operations Forum, Los Angeles, CA.

Ferguson, I. A. (1992). "Touring Machines: Autonomous Agents with Attitudes." Computer 25(5). pp 51-55.

Franklin, S. and A. Graesser (1996). Is it an agent, or just a program? A taxonomy of autonomous agents. Third International Workshop on Agent Theories, Architectures and Languages.

Ghalimi, I. (2002). Design Driven Architecture. Available: http://www.intalio.com/education/notes/note.xpg?id=Design_Driven_Architecture. Accessed: May 2005

Gibson, J. J. (1979). The ecological approach to visual perception. Hillsdale, NJ, Lawrence Erlbaum Associates.

Gibson, J. J. and L. E. Crooks (1938). "A theoretical field-analysis of automobile-driving." The American Journal of Psychology 51(3). pp 453-471.

Hayden, S., C. Carrick, et al. (1999). Architectural Design Patterns for Multiagent Coordination. International Conference on Agent Systems (Agents '99), Seattle, WA.

Hayes, C. C. (1999). "Agents in a nutshell - a very brief introduction." IEEE Transactions on Knowledge and Data Engineering 11(1).

Heineman, G. T. and W. T. Councill (2001). Component Based Software Engineering: putting the pieces together, Addison-Weseley Professional.

Hissam, S., J. Hudak, et al. (2003). Predictable Assembly of Substation Automation Systems: An experiment report, second edition. Carnegie Mellon University, Software Engineering Institute. CMU/SEI-2002TR-031, ESC-TR-2002-031.

Hissam, S. and M. Klein (2004). A model problem for an open robotics controller. Carnegie Mellon University, Software Engineering Institute. CMU/SEI-2004-TN-030.

Holland, O. and C. Melhuish (1999). "Stigmergy, self-organization, and sorting in collective robotics." Artificial Life 5. pp 173-202.

Hollnagel, E. (1993). Human reliability analysis: context and control. London, Academic Press.

Hors, A. L., P. L. Hégaret, et al. (2004). Document Object Model (DOM) Level 3 Core Specification. Available: http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/. Accessed: March 2005

Huhns, M. N. and L. M. Stephens (2000). Multiagent systems and societies of agents. G. Weiss (Ed.). Multiagent Systems: a modern approach to distributed artificial intelligence, The MIT Press.

IAI (2004). CybelePro Agent Infrastructure: User's Guide: Version 1.0. Available: http://www.opencybele.org/docs/UsersGuideCybeleProVersion1.0.pdf. Accessed: January 2005

Ippolito, C. A. and A. R. Pritchett (2000). SABO: A self-assembling architecture for complex system simulation. Proceedings of the 38th AIAA Aerospace Sciences Meeting and Exhibit, Reno, NV.

Jennings, N. R. and M. Wooldridge (2000). Agent-oriented software engineering. 9th European Workshop on Modeling Autonomous Agetns in a Multi-Agent World: Multi-Agent System Engineering (MAAMAW-99).

John, B. E. and D. E. Kieras (1994). The GOMS Family of Analysis Techniques: Tools for Design and Evaluation. Human-Computer Interaction Institute. CMU-HCII-94-106.

Kang, M. (2001). "Team-Soar: a computational model for multilevel decision making." IEEE Transactions on Systems, Man and Cybernetics 31(6). pp 708-714.

Kang, M., L. B. Waisel, et al. (1998). Team-Soar: A model of team decision making. M. J. Prietula, K. M. Carley and L. Gasser (Eds.). Simulating Organizations: computational models of institutions and groups, AAAI Press/The MIT Press.

Kieras, D. E. and D. E. Meyer (1996). The Epic architecture: Principles of operation. Available: ftp://www.eecs.umich.edu/people/kieras/EPIC/EPICArch.pdf. Accessed: September 2005

Law, A. M. and W. D. Kelton (1999). Simulation modeling and analysis. New York, NY, McGraw-Hill.

Lee, S. M. (2002). Agent-based simulation of socio-technical systems: software architecture and timing mechanisms. Industrial and Systems Engineering. Atlanta, GA, Georgia Institute of Technology. PhD.

Lehman, J. F., J. Laird, et al. (1993). A Gentle Introduction to Soar, an Architecture for Human Cognition. Available: http://www.eecs.umich.edu/~soar/sitemaker/docs/misc/Gentle.pdf. Accessed: March 2005

Lewin, K. (1936). Principles of Topological Psychology, McGraw-Hill.

Mann, J., C. Stevenson, et al. (2002). Introducing Time-Based Metering at Los Angeles Air Route Traffic Control Center. Available: http://www.faa.gov/programs/oep/v7/library/Technologies/1/TMA-ATCA%2008-23-02%20km.pdf. Accessed: May 2005

Marrow, A. F. (1969). The Practical Theorist, Basic Books Inc.

Merriam-Webster (2003). Merriam-Webster's Collegiate Dictionary, 11th Edition with CD-ROM and Online Subscription, Merriam-Webster.

Muller, J. P., M. Pischel, et al. (1995). Modelling reactive behaviour in vertically layered agent architectures. M. Wooldridge (Ed.). Intelligent Agents: Theories, Architectures, and Languages. Heidelberg, Germany, Springer Verlag. LNAI 890.

Nair, R., M. Tambe, et al. (2003). Role allocation and reallocation in multiagent teams: Towards a practical analysis. International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2003).

Nair, R., M. Tambe, et al. (2003). Taming Decentralized POMDPs: Towards efficient policy computation for multi-agent settings. International Joint Conference on Artificial Intelligence.

Newell, A. (1990). Unified Theories of Cognition. Cambridge, MA, Harward University Press.

Norman, D. A. (1981). Steps toward a cognitive engineering: System images, system friendliness, mental models. University of California Press.

Ockermann, J. J. and A. R. Pritchett (2000). "A Review and Reappraisal of Task Guidance: Aiding Workers in Procedure Following." International Journal of Cognitive Ergonomics 4(3). pp 191-212.

Odell, J., H. V. D. Parunak, et al. (2002). Modeling agents and their environment. F. Giunchiglia, J. Odell and G. Weiss (Eds.). Agent Oriented Software Engineering (AOSE) III. Berlin, Lecture Notes on Computer Science, Springer. 2585. pp 16 - 31.

Parunak, H. V. D. (2000). Industrial and Practical Applications of DAI. G. Weiss (Ed.). Multiagent systems: a modern approach to distributed artificial intelligence, The MIT Press. pp 377-421.

Parunak, H. V. D., R. Savit, et al. (1998). Agent-based modeling vs. equation-based modeling. Workshop on Modeling Agent Based Systems (MABS98).

Potter, S. S., W. C. Elm, et al. (2002). Using Intermediate Design Artifacts to Bridge the Gap Between Cognitive Analysis and Cognitive Engineering. M. D. McNeese and M. A. Vidulich (Eds.). Cognitive systems engineering in military aviation environments: avoiding cogminutia fragmentosa! Human Systems Information Analysis Center. State of the Art Report.

Rao, A. S. and M. P. Georgeff (1991). Modeling rational agents within a BDI-architecture. Knowledge Representation and Reasoning, San Mateo, CA.

Rasmussen, J. (1976). Outlines of a Hybrid Model of the Process Plant Operator. T. B. Sharidan and G. Johannsen (Eds.). Monitoring behavior and supervisory control. New York, Plenum. pp 371-383.

Rasmussen, J. (1983). "Skills, Rules, and Knowledge; Signals, Signs and Symbols, and Other Distinctions in Human Performance Models." IEEE Transactions on Systems, Man and Cybernetics 13(3).

Rasmussen, J. (1985). "The role of hierarchical knowledge representation in decision making and system management." IEEE Transactions on Systems, Man and Cybernetics 15. pp 234-243.

Rasmussen, J. (1988). Cognitive engineering, a new profession? L. P. Goodstein, H. B. Anderson and S. E. Olsen (Eds.). Task, Errors and Mental Models. London, UK, Taylor and Francis. pp 325 - 334.

Rasmussen, J., A. M. Pejtersen, et al. (1994). Cognitive Systems Engineering, Wiley Series in Systems Engineering.

Rogerson, D. (1997). Inside COM: Microsoft's Component Object Model, Microsoft Press.

Roth, E. M., E. S. Patterson, et al. (2001). Cognitive Engineering: Issues in User-Centered System Design. J. J. Marciniak (Ed.). Encyclopedia of Software Engineering. New York, Wiley-Interscience, John Wiley and Sons.

Rouse, W. B. (1984). Developing an Evaluation Plan: Computer-generated Display System Guidelines. Search Technology Inc. EPRI-NP-3701.

Rouse, W. B., P. R. Frey, et al. (1984). Classification and Evaluation of Decision Aids for Nuclear Power Plant Operators. Search Technology Inc. 8303-1.

Russell, S. and P. Norvig (1995). Artificial Intelligence: a modern approach, Prentice Hall.

Sandholm, T. W. (2000). Distributed Rational Decision Making. G. Weiss (Ed.). Multiagent Systems: A modern approach to distributed artificial intelligence. Cambridge, MA, The MIT Press. pp 201-258.

Schraagen, J. M., S. F. Chipman, et al., Eds. (2000). Cognitive Task Analysis. Mahwah, NJ, Lawrence Erlbaum Associates.

Shah, A. P. and A. R. Pritchett (2005). Agent-based modeling and simulation of socio-technical systems. W. B. Rouse and K. R. Boff (Eds.). Organizational Simulation: From modeling and simulation to games and entertainment. New York, John Wiley.

Shah, A. P. and A. R. Pritchett (2005). Work Environment Analysis: Environment centric multi-agent simulation for design of socio-technical systems. P. Davidsson, B. Logan and K. Takadama (Eds.). Multi-Agent and Multi-Agent-Based Simulation: Joint Workshop MABS 2004. Berlin, Springer Verlag. Lecture Notes In Aritificial Intelligence 3415. pp 65-77.

Shepherd, A. (1998). "HTA as a framework for task analysis." Ergonomics 41(11).

SimAgent The SimAgent Toolkit. Available: http://www.cs.bham.ac.uk/~axs/cog_affect/sim_agent.html. Accessed: July 2005

Singh, M. P., A. S. Rao, et al. (2000). Formal methods in DAI: logic-based representation and reasoning. G. Weiss (Ed.). Multiagent Systems: a modern approach to distributed artificial intelligence, The MIT Press.

Skilton, W., S. Cameron, et al. (1998). Supporting cognitive work analysis with the work domain analysis workbench (WDAW). OzCHI 1998, Adelaide, South Australia.

Smith, B. R. and S. W. Tyler (1997). The Design and Application of MIDAS: A Constructive Simulation for Human-System Analysis. 2nd Simulation Technology & Training (SIMTECT) Conference, Canberra, Australia.

Sperling, B. K. (2005). Information Sharing Strategies to Improve Team Mental Models in Complex Systems. School of Industrial and Systems Engineering. Atlanta, Georgia Institute of Technology. Ph.D.

StarLogo (2004). StarLogo. Available: http://education.mit.edu/starlogo/. Accessed: January 2005

Swarm Swarm. Available: http://www.swarm.org/wiki/Main_Page. Accessed: February 2005

TeamBots (2000). TeamBots. Available: http://www.teambots.org/. Accessed: March 2004

Treisman, A. M. and G. Gelade (1980). "A Feature-Integration Theory of Attention." Cognitive Psychology 12. pp 97-136.

Turing, A. M. (1950). "Computing machinery and intelligence." Mind 59. pp 433-460.

Verma, S. A. and K. M. Corker (2001). Introduction of Context in Human Performance Model as applied to Dynamic Resectorization. 11th International Symposium in Aviation Psychology, Columbus, Ohio.

Vicente, K. J. (1990). "A few implications of an ecological approach to human factors." Human Factors Society Bulletin 33(11). pp 1 - 4.

Vicente, K. J. (1999). Cognitive Work Analysis: Towards Safe, Productive and Healthy Computer Based Work, Lawrence Erlbaum.

Weiss, G., Ed. (2000). Multiagent systems: A modern approach to distributed artificial intelligence, The MIT Press.

Wilensky, U. (1999). NetLogo. Available: http://ccl.northwestern.edu/netlogo/. Accessed: May 2005

Woods, D. D. and E. M. Roth (1988). "Cognitive Engineering: Human Problem Solving with Tools." Human Factors 30(4). pp 415-430.

Woods, D. D. and E. M. Roth (1988). Cognitive Systems Engineering. M. G. Helander, T. K. Landauer and V. P. Prasad (Eds.). Handbook of Human-Computer Interaction. Amsterdam, Elsevier Science B. V., North-Holland.

Wooldridge, M. (2000). Intelligent Agents. G. Weiss (Ed.). Multiagent Systems: a modern approach to distributed artificial intelligence, The MIT Press.

Wooldridge, M. and N. R. Jennings (1995). "Intelligent Agents: Theory and Practice." Knowledge Engineering Review 10(2).

Wray, R., R. Chong, et al. (1994). A Survey of Cognitive and Agent Architectures. Available: http://ai.eecs.umich.edu/cogarch0/. Accessed: May 2005

Yokoo, M. and T. Ishida (2000). Search Algorithms for Agents. G. Weiss (Ed.). Multiagent Systems: A modern approach to distributed artificial intelligence. Cambridge, MA, The MIT Press. pp 165-200.

Yoshikawa, H. (2003). Modeling humans in human-computer interaction. J. A. Jacko and A. Sears (Eds.). The human-computer interaction handbook: Fundamentals, evolving technologies and emerging applications. Mahwah, NJ, Lawrence Erlbaum Associates. Human Factors and Ergonomics. pp 118-146.

Zandt, T. V., H. Colonius, et al. (2000). "A comparison of two response time models applied to perceptual matching." Psychonomic Bulletin & Review 7(2). pp 208-256.